НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
Кафедра автоматики та управління в технічних системах

«До захисту допущено»
Завідувач кафедри
_____ О.І. Ролік
«___»_____2019 р.

Дипломний проект

на здобуття ступеня бакалавра

з напряму підготовки 6.050103 «Програмна інженерія»

на тему: «Система оцінки якості обслуговування в закладах харчування»

Виконав:
студент IV курсу, групи ІТ-51
Супрун Богдан Михайлович _____

Керівник:
Асистент кафедри АУТС Міщенко В.О.
_____

Рецензент:
_____

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2019 рік

Пояснювальна записка
до дипломного проекту
на тему: «Система оцінки якості обслуговування в закладах харчування»

Київ – 2019 рік

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра автоматики та управління в технічних системах

Рівень вищої освіти – перший (бакалаврський)

Напрям підготовки – 6.050103 «Програмна інженерія»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ О.І. Ролік
«___»_____2019 р.

ЗАВДАННЯ
на дипломний проект студенту
Супруну Богдану Михайловичу

1.Тема проекту «Система оцінки якості обслуговування в закладах харчування», керівник проекту Міщенко Володимир Олександрович, асистент кафедри АУТС, затверджені наказом по університету від

«___»_____ 2019 р. №_____

2. Термін подання студентом проекту 18.05.2019

3. Вихідні дані до проекту
_____
_____

4. Зміст пояснювальної записки
_____
_____

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслеників, плакатів, презентацій тощо)
_____
_____-
_____

7. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання дипломного проекту	Термін виконання етапів проекту	Примітка

Студент Б. М. Супрун

Керівник проекту В. О. Міщенко

ВІДОМІСТЬ ДИПЛОМНОГО ПРОЕКТУ

з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
	4	КПІ.ІП-5201. ХХХХХХ.02.81			
	4	КПІ.ІП-5201. ХХХХХХ.02.81			
	4	КПІ.ІП-5201. ХХХХХХ.03.81			
	4	КПІ.ІП-5201. ХХХХХХ.04.51			
	4	КПІ.ІП-5201. ХХХХХХ.04.51			
	3	КПІ.ІП-5201. ХХХХХХ.04.51			
	3	КПІ.ІП-5201. ХХХХХХ.04.51			
	3	КПІ.ІП-5201. ХХХХХХ.04.51			

АНОТАЦІЯ

Пояснювальна записка дипломного проекту складається з трьох розділів, містить 3 таблиці, 4 додатків, 38 рисунків та 10 джерел – загалом 63 сторінок.

Об`єкт дослідження: веб системи, що використовуються оцінки якості обслуговування закладів харчування.

Мета дипломного проекту: розрорблення веб-додатку, вирішення проблем пов’язаних з оцінкою та проектування системи відгуків для закладів харчування, дослідження методів зберігання та використання великих об’ємів даних для швидкого та комфортного перегляду користувачем, розробити чітку структуру шкали оцінки, зручний інтерфейс веб-додатку та впровадити гнучку систему коментарів.

У першому розділі було проведено дослідження існуючих архітектур, дослідження методів зберігання та використання великих об’ємів даних.

У другому розділі було розроблено архітектуру веб застосунку. Побудовано структурну схему класів та діаграму послідовності.

У третьому розділі проведено тестування веб застосунку за розробленим планом тестування. Описано процес тестування.

У додатках наведено: опис програми, схема структурна класів програмного забезпечення, схема структурна послідовності виконання.

КЛЮЧОВІ СЛОВА: ЯКІСТЬ ОБСЛУГОВУВАННЯ, СИСТЕМА, ВЕБ-ЗАСТОСУНОК

ABSTRACT

The explanatory note of the diploma project consists of three sections, contains 3 tables, 4 annexes, 38 figures and 10 sources - a total of 63 pages.

Object of research: Web systems used to assess the quality of catering services.

The purpose of the diploma project: development of a web application, solving the problems associated with the assessment and design of a system for feedbacks, studying the storage and use of large volumes of data for quick and comfortable viewing of the user, develop a clear structure of the scale of assessment, a convenient web interface. -Admin anmplement a flexible comment system.

In the first section, the study of existing architectures, the study of methods for storing and using large volumes of data was conducted.

In the second section, the web application architecture was developed. A structured scheme of classes and sequence diagrams was constructed.

The third section tests the web application for the developed test plan. The testing process is described.

The appendixes contain: description of the program, the diagram of the structural classes of the software, a diagram of the structure of the execution.

KEYWORDS: QUALITY OF SERVICE, SYSTEM, WEB-APPLICATION

ЗМІСТ

ВСТУП

Якість – це запорука успіху. Але це водночас те, що нам в багатьох випадках не вистачає. А якщо розмова йде про їжу та продукти харчування, що ми вживаємо в закладах харчування, то потрібно бути максимально уважними щодо якості обслуговування.

Кожного дня ми спостерігаємо зростання частки бізнесу закладів харчування, починаючи від закладів швидкого харчування і закінчуючи розкішними ресторанами. Очевидно, що швидке зростання кількості закладів харчування, сприяє розвитку конкурентної спроможності та якості цього сектору. З точки зору користувача, стає неможливо знайти ідеальний варіант для нього без відповідної системи оцінки якості закладів харчування. Саме тому доцільно спроектувати та розробити таку систему, яка б могла задовольнити потреби користувачів та допомогти у виборі підходящого місця харчування на ті чи інші життєві ситуації.

Завданням дипломного проекту є вирішення проблем пов’язаних з оцінкою та проектування системи відгуків для закладів харчування, дослідження методів зберігання та використання великих об’ємів даних для швидкого та комфортного перегляду користувачем. Постає завдання розробити чітку структуру шкали оцінки, зручний інтерфейс веб-додатку та впровадити гнучку систему коментарів. Головною унікальною ознакою системи повинен бути механізм обробки інформації, що генерує текстовий файл, який повністю готовий для друку. Так ми зможемо на регулярній основі друкувати власні книги про заклади харчування та усією інформацією щодо їх якості. Система повинна мати локалізацію за країною, щось схоже на “франшизу”. Кожна країна може мати таку систему з відповідним доменом, який вказує на локалізацію. Система повинна бути повністю автоматизованою і зручною в налаштуванні. Так група головних редакторів зможе легко редагувати і заповняти дані про той чи інший заклад харчування. Щоб оцінка якості закладів харчування не була доволі

суб’єктивною та не базувалася на думці одного користувача, система буде мати два головних критерія оцінки якості закладів харчування: оцінка на основі коментарів та рейтингу користувачів системи, оцінка на основі незалежних експертів. На основі побудованих висновків, буде сформована та надрукована книга. Вона повинна бути в електронному вигляді і розміщена в web-додатку системи. Також кожен користувач зможе знайти повну інформацію про той чи інший заклад харчування та його місце розташування. Для кращого підбору закладів харчування повинна бути сформована сторінка веб-додатку з динамічно заповненою картою, користувач зможе відсортовувати результат пошуку за заданими параметрами та за місцем розташуванням.

1 ОГЛЯД ІСНУЮЧИХ АРХІТЕКТУР

1.1 Інформаційна система

Добре продумана інформаційна система – це система, що спирається на послідовну основу, яка підтримує реагування на зміни і, таким чином, підтримує гнучкість організації, коли виникають нові бізнес або адміністративні ініціативи. Фундамент, відомий як інфраструктура інформаційної системи, складається з основних телекомунікаційних мереж, баз даних і сховищ даних, програмного забезпечення, апаратних засобів і процедур, керованих різними фахівцями. З глобалізацією бізнесу інфраструктура організації часто перетинає багато національних кордонів. Створення та підтримка такої складної інфраструктури вимагає широкого планування та послідовної реалізації для розробки стратегічних корпоративних ініціатив, трансформацій, злиття та поглинань. Необхідність у створенні інфраструктури інформаційної системи для створення значущих варіантів майбутнього корпоративного розвитку є основною концепцією архітектурного проектування.

1.2 Архітектура веб-застосувань

1.2.1 Архітектура як поняття

Архітектура програмного забезпечення це процес перетворення таких характеристик програмного забезпечення, як гнучкість, масштабованість, виконуваність, повторне використання та безпека у структуроване рішення, яке відповідає технічним та діловим очікуванням.

Архітектура програмного забезпечення системи зображує організацію або структуру системи і надає пояснення щодо її поведінки. Система являє собою набір компонентів, які виконують певну функцію або набір функцій. Іншими

словами, архітектура програмного забезпечення забезпечує міцний фундамент, на якому можна будувати програмне забезпечення.

Серія архітектурних рішень і компромісів впливає на якість, продуктивність, ремонтопридатність і загальний успіх системи. Невизначення загальних проблем і довгострокових наслідків може поставити систему під загрозу.

Існують численні структури та принципи архітектури високого рівня, які широко використовуються в сучасних системах. Вони часто називають архітектурними стилями. Архітектура системи програмного забезпечення рідко обмежується єдиним архітектурним стилем. Замість цього, комбінація стилів часто складають повну систему.

1.2.2 Дизайн програмного забезпечення

Дизайн програмного забезпечення - це процес концептуалізації вимог програмного забезпечення до реалізації програмного забезпечення. Це початковий етап упродовж життєвого циклу розробки програмного забезпечення - переведення концентрації з проблеми на вирішення.

При розробці програмного забезпечення процес проектування встановлює план, який враховує вимоги користувачів і працює для визначення оптимальних рішень. План повинен визначити найкращий варіант для реалізації передбачуваного рішення.

Дизайн програмного забезпечення включає всі види діяльності, які допомагають у перетворенні програмного забезпечення з специфікації вимог до реалізації.

1.2.3 Типи архітектур веб-додатків

1.2.3.1 Модель Кліент-Сервер

Клієнт - це комп'ютерний апарат або програмне забезпечення, яке має доступ до служби сервера. Сервер часто (але не завжди) розташований на окремому фізичному комп'ютері.

Сервер - це фізичний комп'ютер, призначений для виконання певних операції для задоволення потреб інших комп'ютерів. Залежно від запущеної операції, це може бути файловий сервер, сервер баз даних, домашній медіа-сервер, сервер друку або веб-сервер.

Модель клієнт-сервер також називають структурою мережевих обчислень, оскільки кожен запит і пов'язані з ним послуги розподіляються по мережі. У архітектурі клієнт-сервер, клієнтський комп'ютер надсилає запит даних на сервер через Інтернет, сервер приймає цей запит, обробляє його і доставляє оброблені дані до клієнта. Однією головною особливістю є те, що серверний комп'ютер має можливість керувати одночасно великою кількістю клієнтів. Крім того, один клієнт може підключатися до безлічі серверів на одній мітці часу, де кожен сервер надає інший набір послуг для цього конкретного клієнта (Рисунок 1.1).

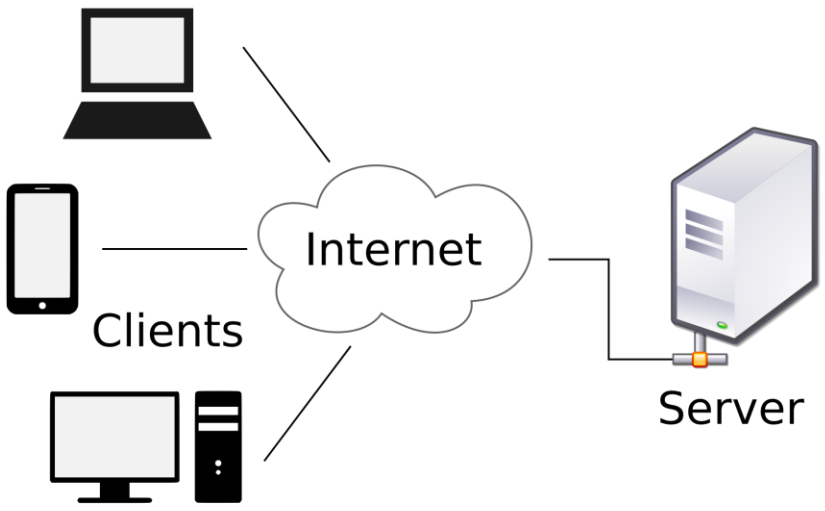

Рисунок 1.1 - Модель Кліент-Сервер [1]

Але для архітектури клієнт-сервер в Інтернеті необхідно враховувати конкретні фактори:

1) Конкретний набір мов разом із стандартом зв'язку, протоколом для взаємодії двох систем. Найбільш популярними є HTTP і HTTPS (Hyper Text Transfer Protocol Secure).
2) Механізм і протокол для запиту необхідних аспектів з сервера. Це може бути в будь-якій структурі форматованих даних. В основному реалізовані і популярні формати виконуються в XML і JSON.
3) Механізм і протокол для відповіді сервера, надіславши відповідь у структуру відформатованих даних (зазвичай XML або JSON).

1.2.3.2 Дизайн шаблону MVC

Model-View-Controller (MVC) є шаблонним дизайном, який розділяє додаток на три основні логічні компоненти: модель, інтерфейс і контролер (Рисунок 1.2). Кожен з цих компонентів побудований для обробки конкретних аспектів роботи програми. MVC є одним з найбільш часто використовуваних галузевих стандартів веб-розробки для створення масштабованих і розширюваних проектів.

Модель - це дані та бізнес-логіка. Модель являє собою форму даних і бізнес-логіку. Він зберігає дані програми. Модельні об'єкти отримують і зберігають стан моделі в базі даних.

Інтерфейс - це UI користувача. Перегляд відображення даних за допомогою моделі для користувача і також дозволяє їм змінювати дані.

Контролер обробляє запит користувача. Як правило, користувач взаємодіє з View, що в свою чергу піднімає відповідний запит URL, цей запит буде оброблений контролером. Контролер надає відповідний вигляд даних моделі як відповідь.

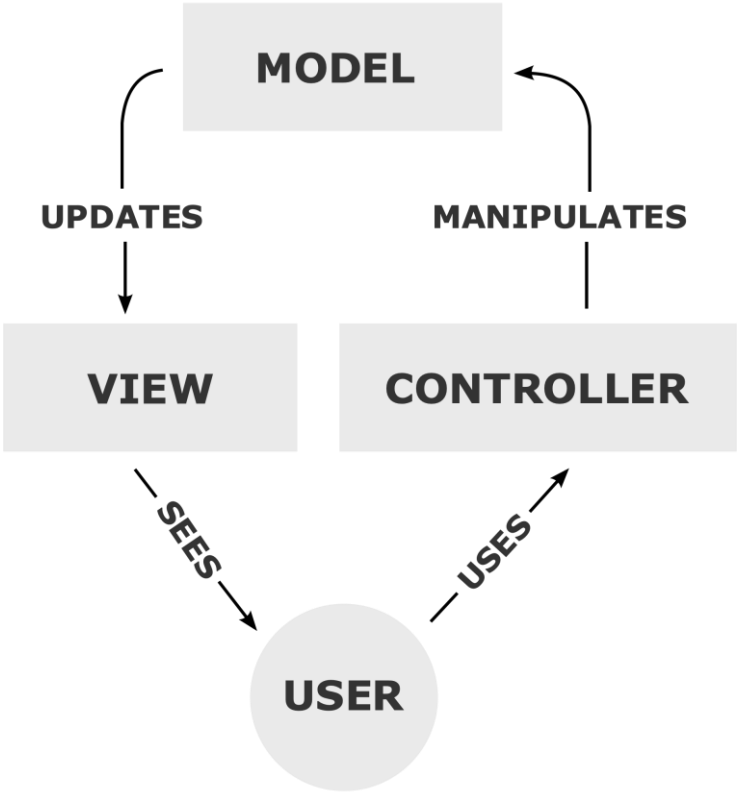

Рисунок 1.2 - Дизайн шаблону MVC [2]

1.2.3.3 Типи архітектури веб-додатків

Створення нового продукту - це ризик. Вибір правильної архітектури є важливим кроком до успіху. Тому розглянемо монолітну, сервісно-орієнтовану, мікросервісну та безсерверну архітектуру веб-додатків:

1) Монолітна архітектура:

Моноліт - древнє слово, яке посилається на величезний камінь. Незважаючи на те, що цей термін широко використовується сьогодні, сутність залишається незмінним для всіх варіантів вживання. В інженерії програмного забезпечення монолітний підхід відноситься до однієї неподільної одиниці. Концепція монолітного програмного забезпечення полягає в різних компонентах програми, об'єднаних в єдину програму на одній платформі. Як правило, монолітний додаток складається з бази даних, клієнтського інтерфейсу користувача та

серверного додатку. Всі частини програмного забезпечення уніфіковані, а всі його функції керуються в одному місці. Давайте детально розглянемо структуру монолітного програмного забезпечення (Рисунок 1.3).

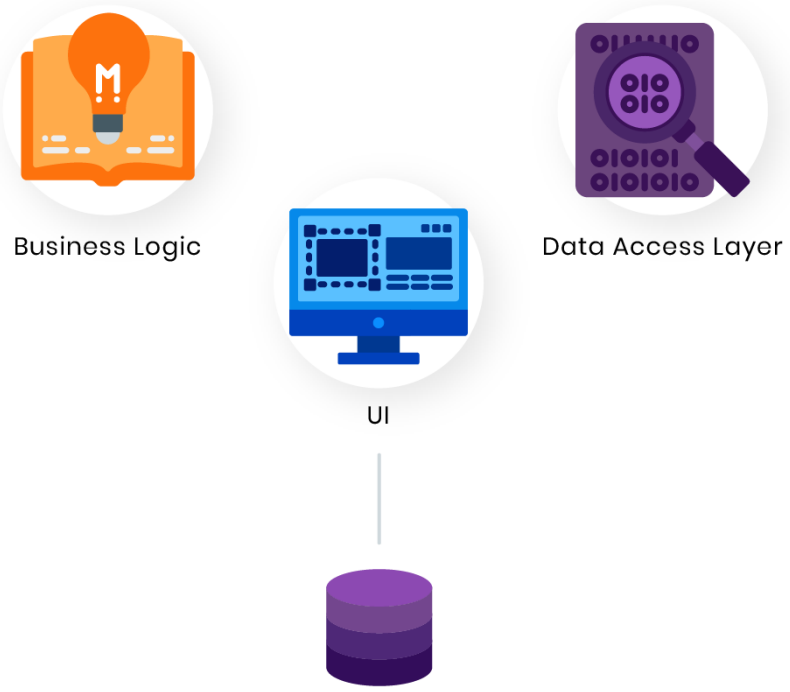

Рисунок 1.3 - Монолітна архітектура [4]

Монолітна архітектура зручна для роботи з маленькими командами, тому багато стартапів вибирають такий підхід при створенні програми. Компоненти монолітного програмного забезпечення взаємопов'язані і взаємозалежні, що допомагає програмному забезпеченню бути автономним. Ця архітектура є традиційним рішенням для побудови додатків, але деякі розробники вважають її застарілою. Однак, я вважаю, що монолітне архітектура є ідеальним рішенням в деяких обставинах.

Щоб з'ясувати, чи є це архітектурне рішення корисним для вашого бізнесу, розглянемо його плюси і мінуси.

Плюси монолітної архітектури:

- Простіша розробка та розгортання: є багато інструментів, які можна інтегрувати для сприяння розвитку. Крім того, всі дії виконуються з одним каталогом, який забезпечує легше розгортання. З монолітним ядром розробникам не потрібно розгортати зміни або оновлення окремо, оскільки вони можуть зробити це одночасно і заощадити багато часу.
- Легка інтеграція систем моніторингу: більшість додатків залежать від великої кількості різнопланових проблем, таких як аналіз трафіку, логування, обмеження швидкості тощо. Монолітні програми значно полегшують ці проблеми завдяки своїй єдиній кодовій базі. Легше підключати компоненти, коли все працює в тому ж самому додатку.
- Краща продуктивність: якщо вони побудовані належним чином, монолітні програми зазвичай є більш ефективними, ніж програми на основі мікросервісу. Наприклад, програма з архітектурою мікросервісів може потребувати 40 викликів API для 40 різних мікросервісів, наприклад, для завантаження кожного екрану, що, очевидно, призводить до більш повільної продуктивності. Монолітні програми, у свою чергу, дозволяють швидше спілкуватися між компонентами програмного забезпечення через спільний код і пам'ять.

Мінуси монолітної архітектури

- Кодова база стає громіздким з часом: з часом більшість продуктів розвивається і збільшується, а їх структура стає розмитою. Кодова база починає виглядати дійсно масивно і стає важко зрозумілою і важко модифікованою, особливо для нових розробників. Також важче знайти побічні ефекти і залежності. Зі зростанням кодової бази інтегроване середовище розробки (IDE) перевантажується.

– Важко прийняти нові технології: якщо потрібно додати нову технологію до вашого додатка, розробники можуть зіткнутися з перешкодами для прийняття. Додавання нових технологій означає перезапис всієї програми, що є дорогим і трудомістким.
– Обмежений деплоймент: у монолітних програмах кожне невелике оновлення вимагає повного розгортання системи на сервері. Таким чином, всі розробники повинні чекати, поки це буде зроблено. Коли кілька команд працюють над одним проектом, швидкість розробки може бути значно зменшена.

Висновок: Монолітна модель не застаріла, і в деяких випадках вона все ще працює. Деякі гігантські компанії залишаються монолітними, незважаючи на сьогоднішню популярність мікросервісів. Монолітна архітектура програмного забезпечення може бути корисною, якщо ваша команда перебуває на стадії створення продукту, ви створюєте докінця не визначений продукт, і у вас немає досвіду роботи з мікросервісами. Монолітний тип архітектури програмного забезпечення ідеально підходить для стартапів, які повинні мати готовий продукт і запустити його роботу як можна швидше.

2) Сервіс-орієнтована архітектура:

Сервіс-орієнтована архітектура (SOA) - це стиль архітектури програмного забезпечення, який відноситься до програми, що складається з дискретних і слабо пов'язаних програмних агентів, які виконують необхідну функцію. SOA має дві основні ролі: постачальник послуг і споживач послуг. Обидві ці ролі можуть бути відтворені програмним агентом. Концепція SOA полягає в наступному: додаток може бути спроектований і побудований таким чином, що його модулі легко інтегруються і можуть бути легко повторно використані.

Плюси сервіс-орієнтованої архітектури:

– Повторне використання послуг: завдяки самодостатній і слабо пов'язаній природі функціональних компонентів у сервісно-

орієнтованій архітектурі ці компоненти можуть бути повторно використані в багатьох інших додатках без впливу на інші служби.

- Краща ремонтопридатність: оскільки кожна послуга програмного забезпечення є незалежною одиницею, її легко оновлювати та підтримувати, не зашкоджуючи іншим службам. Наприклад, керувати великими корпоративними програмами можна легше, якщо вони розбиті на служби.
- Вища надійність: сервіси легше налагоджувати і тестувати, ніж величезні шматки коду, як у монолітному підході. Це, в свою чергу, робить продукти SOA більш надійними.
- Паралельний розвиток: оскільки сервіс-орієнтована архітектура складається з шарів, вона виступає за паралелізм у процесі розробки. Незалежні послуги можуть бути розроблені паралельно і одночасно завершені.

Мінуси сервіс-орієнтованої архітектури:

- Комплексне управління: головним недоліком сервісно-орієнтованої архітектури є її складність. Кожна служба повинна переконатися, що повідомлення доставляються вчасно. Кількість цих повідомлень може бути понад мільйон за один раз, що робить велику проблему для управління всіма послугами.
- Високі інвестиційні витрати: розвиток сервіс-орієнтованої архітектури вимагає великих авансових інвестицій людських ресурсів, технологій і розвитку.
- Додаткове перевантаження: у сервіс-орієнтованої архітектури всі входи перевіряються, перш ніж одна служба взаємодіє з іншою службою. При використанні декількох служб це збільшує час відгуку та зменшує загальну продуктивність.

Висновок: Підхід сервіс-орієнтованої архітектури найкраще підходить для складних корпоративних систем, таких як банки. Банківську систему надзвичайно важко розбити на мікросервіси. Але монолітний підхід також не є корисним для банківської системи, оскільки одна частина може зашкодити цілому додатку. Найкраще рішення - використовувати підхід сервіс-орієнтованої архітектури і організувати складні програми в ізольованих незалежних службах.

3) Мікросервісна архітектура:

Мікросервіс - це тип архітектури програмно-орієнтованого програмного забезпечення, який зосереджується на створенні ряду автономних компонентів, що складають додаток. На відміну від монолітних додатків, побудованих як єдина неподільна одиниця, програми побудовані на мікросервісній архітектурі складаються з декількох незалежних компонентів, які спілкуються через API (Рисунок 1.4).

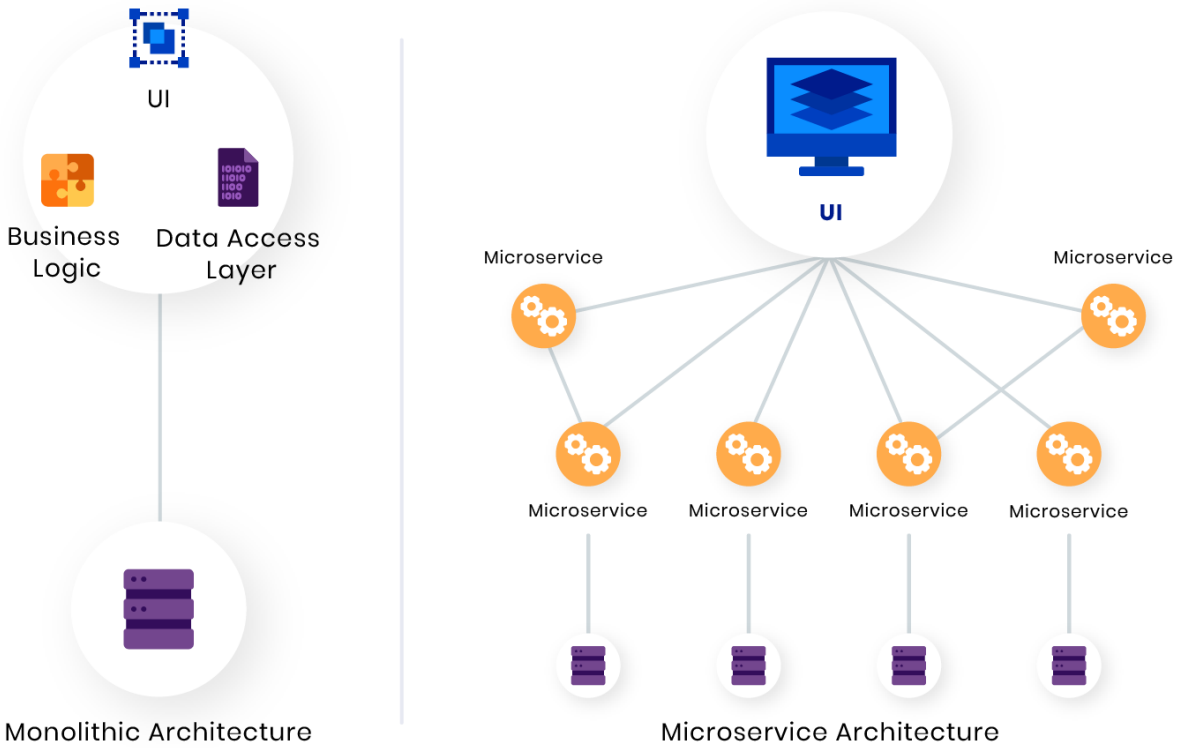

Рисунок 1.4 - Мікросервісна і монолітна архітектура [4]

Підхід мікросервісів зосереджений переважно на бізнес-пріоритетах і потенційних можливостях, тоді як монолітний підхід організований навколо технологічних шарів, інтерфейсів та баз даних. Підхід мікросервісів став тенденцією останніх років, оскільки все більше підприємств стають гнучкими і рухаються до DevOps технологіям.

Є багато прикладів компаній, які розвивалися від монолітного підходу до мікросервісів. Серед найбільш відомих є Netflix, Amazon, Twitter, eBay і PayPal. Щоб визначити, чи підходять мікрослужби для нашого проекту, давайте визначимо плюси і мінуси цього підходу.

Плюси мікросервісів:

- Легко розробляти, тестувати і розгортати: найбільша перевага мікросервісів над іншими архітектурами програмного забеспечення полягає в тому, що невеликі поодинокі послуги можуть бути побудовані, перевірені і розгорнуті незалежно. Оскільки блок розгортання невеликий, він полегшує і прискорює розробку і випуск. Крім того, випуск однієї одиниці не обмежується випуском ще не завершеного пристрою. І останнім плюсом тут є те, що ризики розгортання скорочуються, оскільки розробники розгортають частини програмного забезпечення, а не весь додаток.
- Підвищена швидкість розробки: за допомогою мікросервісів кілька команд можуть працювати над своїми сервісами самостійно і швидко. Кожна окрема частина програми може бути побудована незалежно від інших компонентів системи. Наприклад, у вас може бути команда з 100 осіб, яка працює над цілим додатком (як у монолітному підході), або ви можете мати 10 команд з 10 осіб, які розробляють різні сервіси для програми. Підвищена швидкість розробки дозволяє розробникам оновлювати компоненти системи, не розгортаючи всю програму на сервері. Більш того, гнучкість забезпечує більш безпечний процес

розгортання та поліпшення роботи. Нові функції можна додавати за потреби, не чекаючи запуску всієї програми.

– Можливість масштабування по горизонталі: вертикальне масштабування (запуск одного й того ж програмного забезпечення, але на більших машинах) може бути обмежене можливостями кожної служби. Але горизонтальне масштабування (створення більшої кількості служб в одному пулі) не обмежене і може працювати динамічно за допомогою мікросервісів. Крім того, горизонтальне масштабування може бути повністю автоматизованим.

Мінуси мікросервісів:

– Складність: найбільшим недоліком мікросервісів є їхня складність. Розбиття програми на незалежні мікросервіси тягне за собою більшу кількість компонентів. Такий тип архітектури вимагає ретельного планування, величезних зусиль, ресурсів команди і навичок.

– Проблеми безпеки: у додатку мікросервісів кожна функціональність, яка здійснює зовнішній зв'язок через API, збільшує ймовірність атак. Ці атаки можуть відбуватися лише за умови, що при створенні програми не буде виконано належних вимірювань безпеки.

– Різні мови програмування: можливість вибору різних мов програмування - це дві сторони однієї медалі. Використання різних мов ускладнює розгортання. Крім того, важче керувати програмістами між етапами розробки, коли кожна служба написана іншою мовою.

Висновок: Мікросервісну архітектуру потрібно використовувати не для всіх типів програм. Ця модель ідеально підходить для розвитку додатків і складних систем. Розглядати можливість вибору архітектури мікросервісів можна коли є декілька досвідчених команд і коли додаток є достатньо складним, щоб розбити його на служби. Коли програма велика і має бути гнучкою і масштабованою, мікросервісна архітектура буде вигідна.

4) Архітектура без серверів:

Архітектура без серверів - це хмарний обчислювальний підхід до побудови та запуску програм і служб без необхідності управління інфраструктурою. У безсерверних додатках виконання коду управляється віддаленим сервером, що дозволяє розробникам розгортати код, не турбуючись про обслуговування та забезпечення сервером (Рисунок 1.5). Насправді, без серверів це не означає «немає сервера». Додаток все ще працює на серверах, але сторонні хмарні служби, такі як AWS, беруть на себе повну відповідальність за ці сервери. Архітектура без серверів позбавляє від необхідності додаткових ресурсів, масштабування додатків, обслуговування серверів, систем баз даних і сховищ.

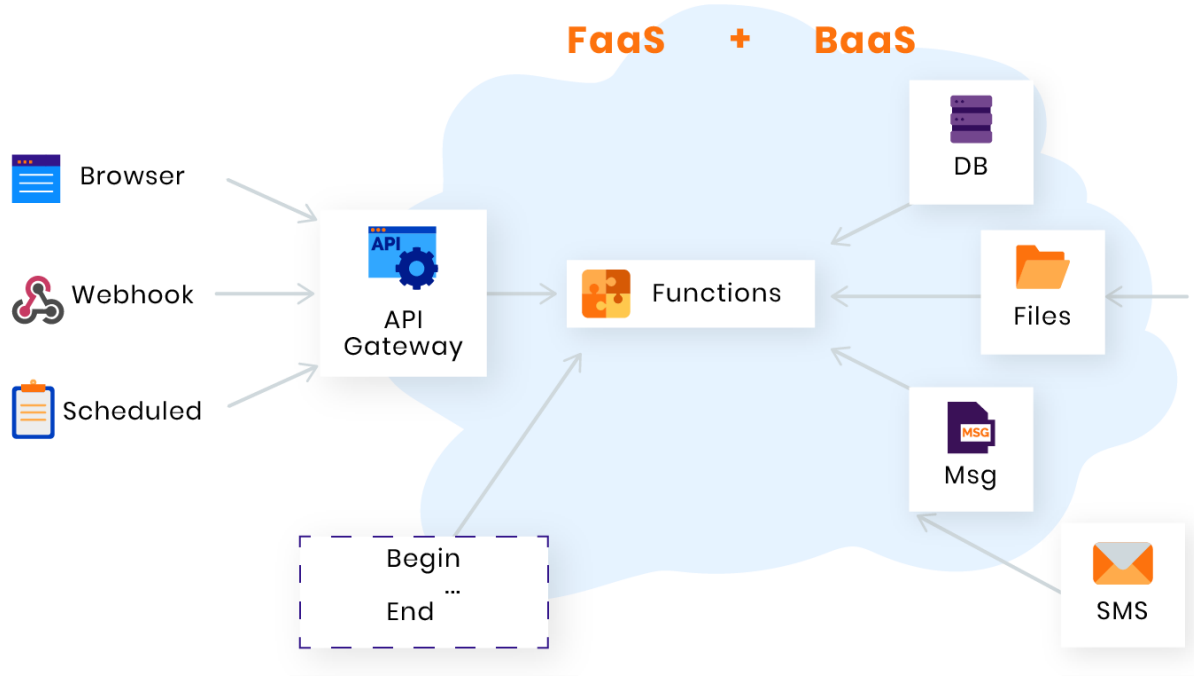

Рисунок 1.5 - Архітектура без серверів [4]

Архітектура без серверів включає дві концепції:

– FaaS (функція як послуга) - модель хмарних обчислень, яка дозволяє розробникам завантажувати фрагменти функціональності в хмару і дозволяти виконувати їх окремо

– BaaS (бекенд як послуга) - модель хмарних обчислень, яка дозволяє розробникам передавати зовнішні аспекти (керування базами даних, хмарне зберігання, хостинг, аутентифікація користувачів тощо) і записувати та підтримувати лише частину інтерфейсу.

Використовуючи безсерверну архітектуру, розробники можуть зосередитися на самому продукті, не турбуючись про управління сервером або середовищами виконання. Це дозволяє розробникам зосередитися на розробці продуктів з високою надійністю і масштабованістю.

Плюси безсерверної архітектури

– Простота розгортання: у безсерверних програмах розробникам не потрібно турбуватися про інфраструктуру. Це дозволяє їм зосередитися на самому коді. Архітектура без серверів дозволяє дуже швидко розгортати програму, оскільки розгортання займає лише години або дні (порівняно з днями або тижнями з традиційним підходом).
– Зниження витрат: перехід на без серверну архітектуру знижує витрати. Оскільки вам не потрібно керувати базами даних, певною логікою та серверами, ви можете не тільки створювати більш якісний код, але й скорочувати витрати. Використовуючи модель без сервера, ви стягуєте плату лише за цикли процесора та пам'ять, які ви фактично використовуєте.
– Покращена масштабованість: багато власників бізнесу хочуть, щоб їхні програми стали впливовими та масштабованими, як Google або Facebook. Сервернеобчислювальне обчислення робить автоматичне та безшовне масштабування. Ваша програма автоматично масштабуватиметься, відповідно від завантаження програми або база користувачів зростатиме, не впливаючи на продуктивність. Серверні програми можуть обробляти величезну кількість запитів, тоді як традиційний додаток буде перевантажений раптовим збільшенням запитів.

Мінуси безсерверної архітектури

- Прив’язка до постачальника: ви надаєте постачальнику повний контроль над вашими операціями. Як наслідок, зміни в бізнес-логіці обмежені, а міграція від одного постачальника до іншого може бути непростою.
- Не для довгострокових завдань: модель без серверів не підходить для довгострокових операцій. Програми без серверів добре підходять для коротких процесів у реальному часі, але якщо завдання займає більше п'яти хвилин, знадобиться додаткова функціональність FaaS.

Висновок: Архітектура безсерверного програмного забезпечення вигідна для виконання одноразових завдань і допоміжних процесів. Вона відмінно працює для важких для клієнта програм і додатків, які швидко зростають і потребують безмежного масштабування.

1.2.4 Моделі взаємодії клієнта і сервера

Веб-сервіси є ключовою точкою інтеграції для різних додатків, що належать до різних платформ, мов, систем. Веб-сервіси - це набір платформних незалежних API (функцій), які можна використовувати з віддаленого сервера через Інтернет. Існують в основному дві сторони, які беруть участь у цьому, один з яких надає набір відкритих API, а інший, звичайно відомий як споживач веб-сервісів, є стороною, яка використовує функціональні можливості та послуги, що надаються стороною веб-сервісів.

Існують різні способи надання веб-сервісів, але найбільш поширеними є SOAP, XML-RPC і REST:

1) XML-RPC є специфікаціяі набір реалізацій, що дозволяють працювати на різних операційних системах, працюючи в різних середовищах, щоб здійснювати виклики процедур через Інтернет. Це віддалена

процедура, яка використовує HTTP як транспортний протокол передачі даних, а XML - як кодування. XML-RPC розроблений настільки просто, наскільки це можливо, дозволяючи передавати, обробляти і повертати складні структури даних. Він не ставить за мету вирішення кожної проблеми. Натомість він прагне бути простим та ефективним засобом для запиту та отримання інформації. Він використовує XML для кодування і декодування віддаленого виклику процедури разом з його параметрами.

2) SOAP (Simple Object Access Protocol) це спосіб обміну повідомленнями на основі XML через Інтернет для забезпечення та споживання веб-сервісів. Повідомлення SOAP передаються формуванням SOAP-конверта. SOAP широко критикується за складність проекту. SOAP - це легкий протокол для обміну інформацією в децентралізованому, розподіленому середовищі. Це протокол на основі XML, який складається з трьох частин: конверт, який визначає рамки для опису того, що знаходиться в повідомленні, і як його обробляти, набір правил кодування для вираження примірників типів даних, визначених прикладною програмою, і конвенція для представлення виклики та відповіді віддалених процедур.
3) REST - репрезентативна передача стану або REST в основному означає, що кожна унікальна URL-адреса є поданням деякого об'єкта. Ви можете отримати вміст цього об'єкта за допомогою HTTP GET, після чого ви можете використовувати POST, PUT або DELETE для зміни об'єкта (на практиці більшість служб використовують POST для цього). Не потрібно використовувати XML як формат обміну даними в REST. Повернутий формат інформації може бути у форматі XML, JSON, звичайному тексті або навіть у форматі HTML. REST архітектура в основному орієнтована на дві речі: ресурси та інтерфейс. Вона покладається на протокол HTTP для всіх операцій CRUD: створення,

читання, оновлення та видалення. Ресурси - це стан і функціональність програми, яка представлена унікальною URL-адресою. Ресурси поділяють єдиний інтерфейс для передачі стану між клієнтом і сервером. Наприклад, URL, http://example.com/item/11 може бути ресурсом. Припустимо, що метод GET використовується для отримання деталей продукту з цієї URL-адреси, метод POST використовується для зміни інформації про виробництво, а метод DELETE може бути використаний для видалення продукту з того ж URL-адреси. Тут методи HTTP працюють як інтерфейс для доступу до ресурсів. Всі ресурси реалізують один і той же єдиний інтерфейс. Стандартні методи - в даному випадку HTTP-дієслова - відображаються в семантиці ресурсів.

1.3 Огляд існуючих рішень на ринку систем оцінки якості послуг

Поставлена задача полягає в повній автоматизації процесу оцінки якості обслуговування закладів харчування. На даний момент існує ряд програмних рішень, які в певній мірі реалізовують поставлену задачу. Кожне з існуючих рішень реалізовує частину задачі. Частина існуючих рішень спеціалізується тільки на закладах харчування, деякі ні. Нижче наведені найбільш функціональні програмні рішення.

1.3.1 Michelin Guide

Ресторанний гід, що оцінює заклади харчування в особливій оцінці, яка може коливатися від однієї зірки до трьох. Система має всі необхідні складові, для комфортного користування та пошуку закладів харчування (Рисунок 1.6). Цю систему можна назвати “еталонною” та гарним прикладом для поставленої

задачі дипломного проекту. В ній є всі необхідні ознаки бути найкращою альтернативною системою для поставленої задачі дипломного проекту. Присутня динамічна карта з маркерами закладів харчування. Так як Michelin Guide випускає на регулярній основі книги-довідники, можна припустити допущення, що система має механізм генерації книг, подібно до механізму поставленого в завданні дипломного проекту. Доступ до такого механізму через інтерфейс web-додатку відсутній. Тому можна зробити тільки припущення про його існування.

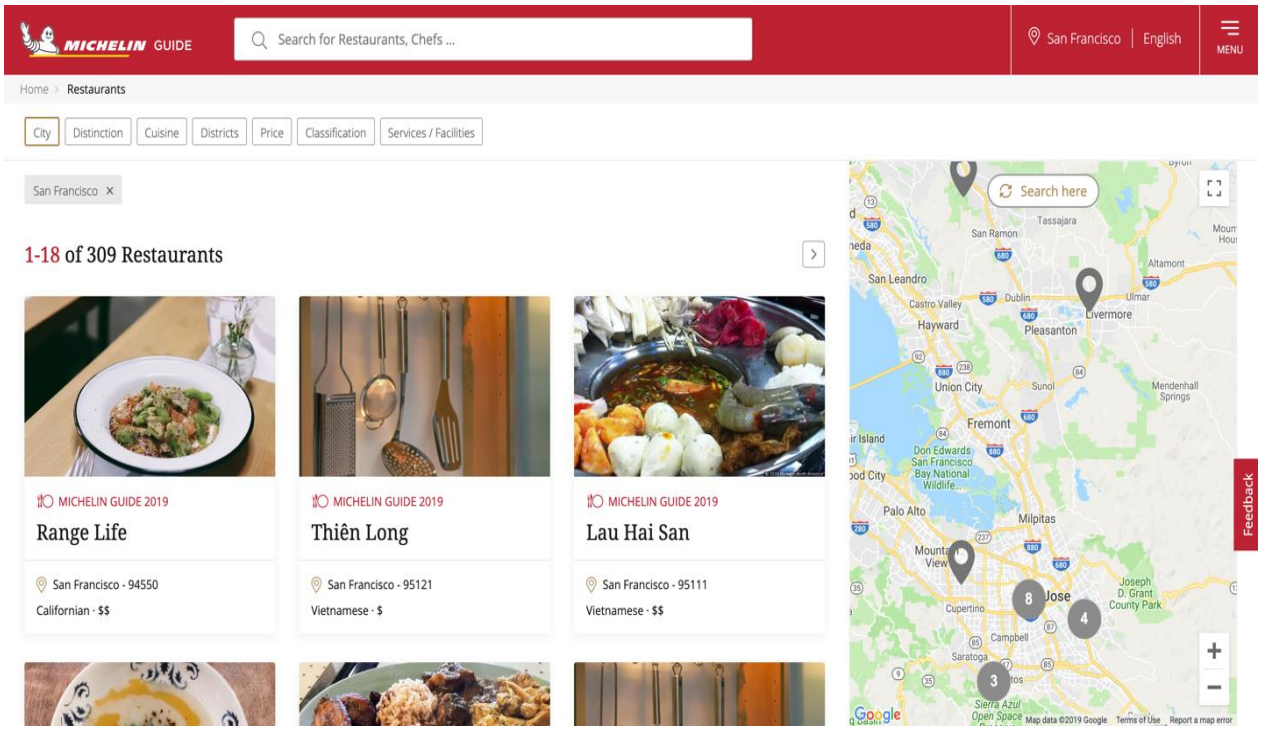

Рисунок 1.6 - Інтерфейс Michelin Guide

Водночав Michelin Guide є елітним ресторанним гідом. Оцінка обслуговування закладів харчування відбувається тільки на основі висококваліфікованих експертів з Michelin Guide, тому основна маса закладів харчування є доволі дорогою для пересічного громанянина тої чи іншої держави. Перелік закладів харчування досить низький. Також система не має можливості доповнення інформації користувачами, додаток дає змогу ознайомитись з

оцінкою, яку залишили самі екперти, що робить продукт менш “народним”. Сама система і критерії оцінювання не відомі і зараз.

1.3.2 Yelp

Веб-сайт для пошуку на місцевому ринку послуг, наприклад ресторанів або перукарень, з можливістю додавати та переглядати рейтинги та оцінки цих послуг. Для популярних бізнесів є сотні оцінок. Для оглядачів на сайті передбачені елементи соціальної мережі (Рисунок 1.7).

Дуже популярний цей ресурс на території США та включає в собі не тільки оцінки обслуговування закладів харчування. Сервіс має зручний інтерфейс, динамічну карту з маркерами запропонованих послуг, досить продуману і інтуїтивно зрозумілу систему написання відгуку до тої чи іншої запропонованої послуги. Кожен користувач може залишити всій відгук, який буде складатись не тільки з тексту, а й фотографій, рейтингу оцінки від 1 до 5, категорії запропонованої послуги та багато іншого. Система відгуків є доволі схожою на соціальну мережу, користувачі можуть коментувати відгуки, можуть ставити рейтинг на відгуках, задавати питання та спілкуватися щодо відгуків, меню, фотографій.

Yelp може стати гарним прототипом для системи відгуків на основі звичайних користувачів для дипломного проекту. Широкий функціонал дозволяє як найкраще передати враження щодо закладу харчування. Водночас рейтинг будується тільки на основі відгуків користувачів, що може спричинити деякі проблеми з достовірністю відгуків. Ресурс є доступний кожному, тому існує ймовірність сфальсифікованих відгуків на заклади харчування. Також сайт охоплює велику кількість категорій в сфері бізнесу обслуговування, що може шкодити концентрації користувачів. Тому що різноманітність пропонованих

послуг є дуже різкою, починаючи з перукарень та закінчуючи послугами заміни масла в автомобілях.

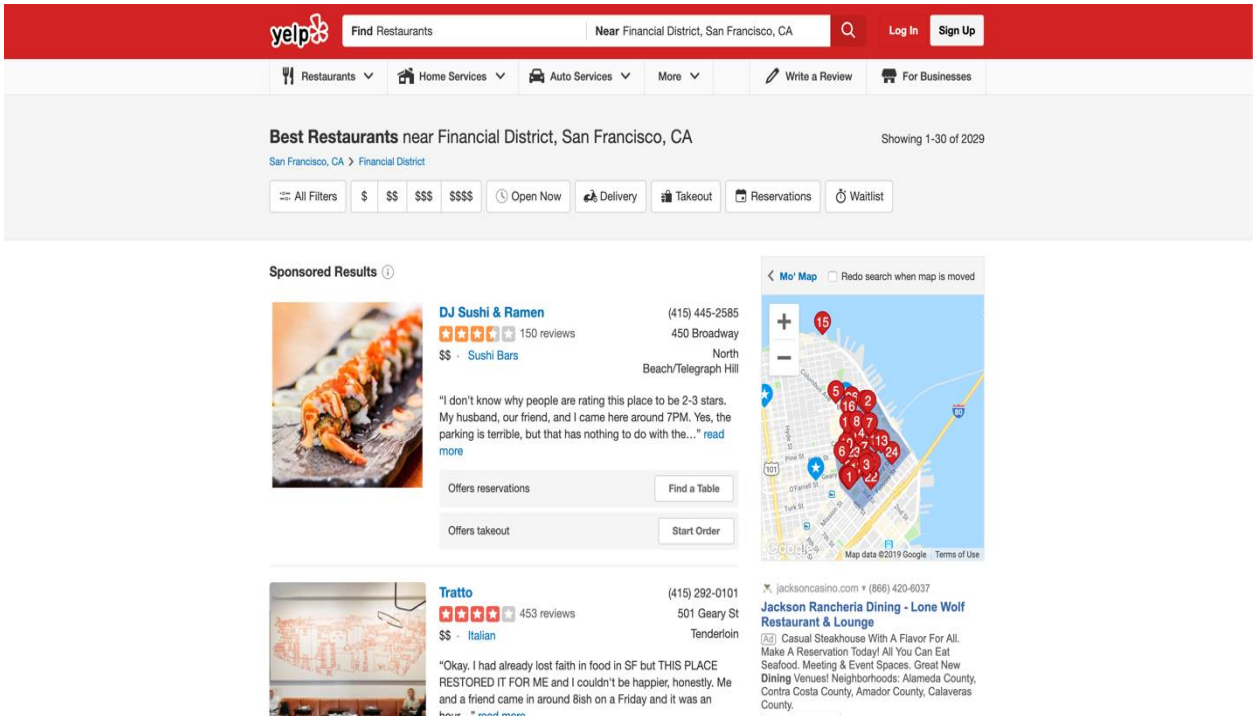

Рисунок 1.7 - Інтерфейс Yelp

1.3.3 TripAdvisor

Американський веб-сайт, що дозволяє своїм користувачам спланувати майбутню подорож до будь-якої країни світу. Послуги сайту є безкоштовні для користувачів, які надають більшу частину контенту. За даними офіційного веб-сайту компанії, TripAdvisor щомісяця обслуговує 315 мільйонів унікальних відвідувачів, більш ніж 70 мільйонів зареєстрованих користувачів, які залишили більше ніж 200 мільйонів відгуків. Схожий на попередній варіант – гігант, як включає не тільки оцінки обслуговування закладів харчування (Рисунок 1.8).

Міста та готелі, ресторани та атракціони зберігаються в базі даних Advisor Trip, що робить його корисним при плануванні багатьох аспектів відпочинку. Індекс популярності готелю TripAdvisor надає рейтинг найкращих готелів міста

і може бути відфільтрований за рейтингом зірок. Якщо є якісь питання, то майже 2/3 повідомлень на форумі Trip Advisor отримують відповідь протягом 24 годин. Відкриті фотографії, які постійно оновлюються, членів TripAdvisor надають нові перспективи. Безкоштовні програми TripAdvisor доступні для мобільних телефонів і планшетів.

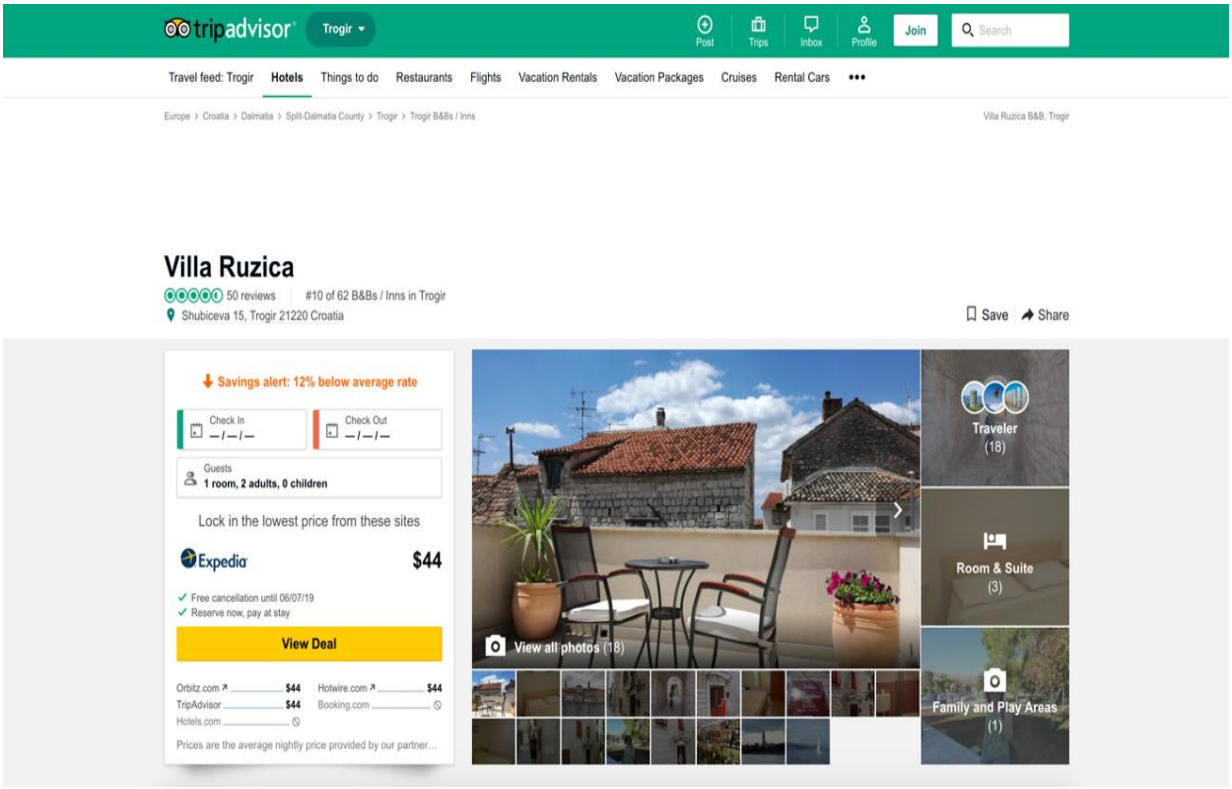

Рисунок 1.8 - Інтерфейс TripAdvisor

1.3.4 Google Maps

Безкоштовний картографічний сервіс від компанії Google, а також набір додатків, побудованих на основі цього сервісу й інших технологій Google. Сервіс являє собою карту та супутникові знімки всього світу і надає користувачам можливості панорамного перегляду вулиць (Google Street View), аналізу трафіку у реальному часі (Google Traffic), прокладання маршруту (автомобілем, пішки, велосипедом або громадським транспортом) (Рисунок 1.9). З сервісом

інтегрований бізнес-довідник і карта автомобільних доріг, з пошуком маршрутів. Бізнес-довідник являє собою короткі відомості про заклад харчування або інший заклад обслуговування. Є відомості про розташування закладу, можна з легкістю прокласти маршрут до нього (автомобілем, пішки, велосипедом або громадським транспортом). Бізнес-довідник також має фотографіїї, що залишили користувачі та відгуки. Користувачі сервісу самі залишають відгуки та рейтинги для закладів харчування або інший закладів обслуговування.

Основний напрям Google Maps – це карти, що дуже зручно для швидкого знаходження місця розташування закладу харчування. Також сервіс не розкриває повної інформації про заклад харчування так як присутня тільки назва закладу, його адреса, посилання на web-сайт, якщо він є, розклад роботи і в деяких випадках меню. Система відгуків доволі проста та задовольняє мінімальні потреби. Робота з картою насичена різноманітним функціоналом, що буде корисним для дипломного проекту.

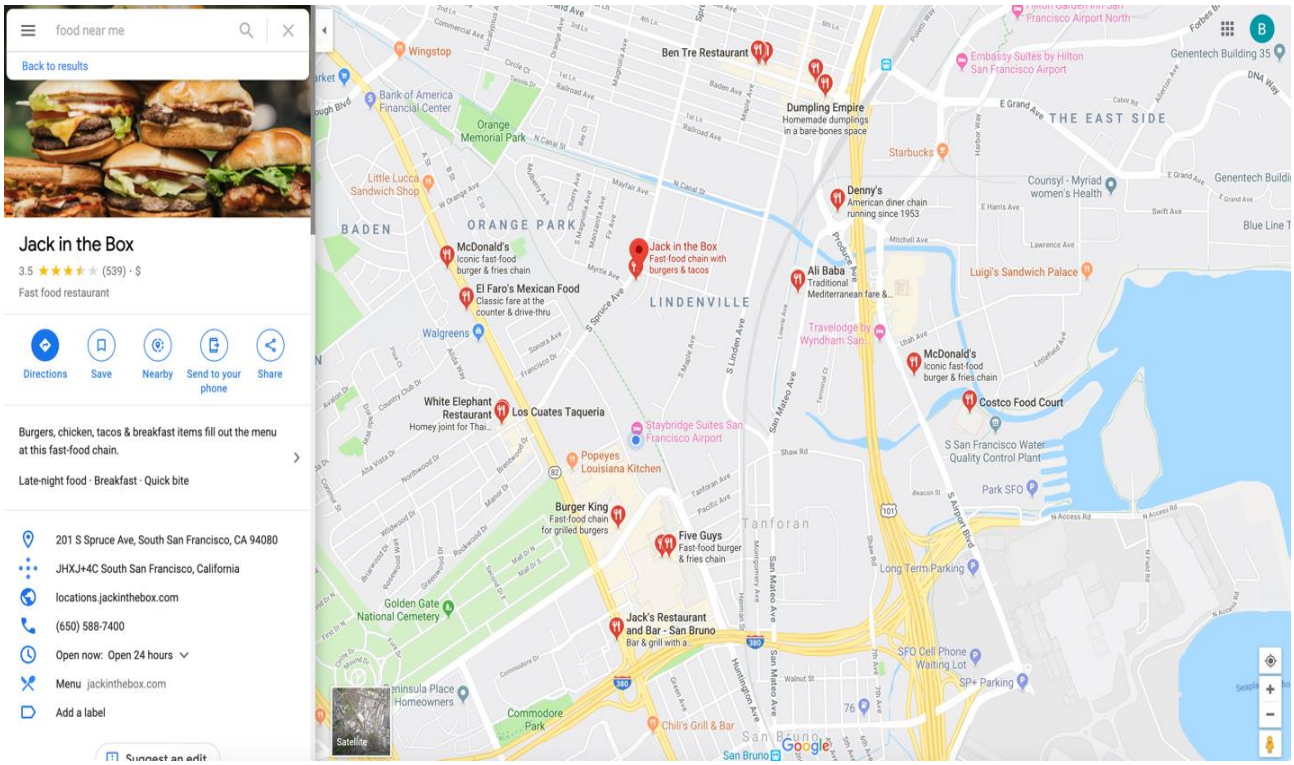

Рисунок 1.9 - Інтерфейс Google Maps

1.4 Висновки

В розділі були розглянуті основні поняття, типи клієнт-серверних архітектур, моделі передачі даних, їх переваги та недоліки. Після дослідження та аналізу переваг і недоліків, було зроблено висновок, що для системи оцінки якості обслуговування в закладах харчуваннях є доцільним обрання монолітний тип архітектури. Монолітна архітектура зручна для роботи з маленькими командами, тому було вибрано такий підхід для створення дипломного проекту. Компоненти монолітного програмного забезпечення будуть взаємопов'язані і взаємозалежні, що допоможе програмному забезпеченню бути автономним. Є багато інструментів, які можна інтегрувати для сприяння розвитку. Для моделі передачі даних було обрано архітектурний стиль REST. Він дозволяє використовувати більшу різноманітність форматів даних, наприклад JSON. У поєднанні з JSON (який, як правило, працює краще з даними та забезпечує швидший аналіз), REST, як правило, вважається більш зручним для роботи. REST забезпечує чудову продуктивність, зокрема, через кешування інформації, яка не змінена і не динамічна.

Коли продукт не є унікальним на світовій платформі, то побудова архітектури не можлива без огляду існуючих рішень. Після потрібно вибрати сильні сторони конкурентів та реалізувати в дипломному проект. Так Michelin Guide має своїх професійних експертів, які визначають рейтинг закладів харчування. Система зосереджена тільки на закладах харчування. Yelp пропонує рейтинг різноманітних послуг, який був визначений самими користувачами. Система відгуків є доволі схожою на соціальну мережу, що дає можливість краще передати враження щодо закладу харчування. Google Maps є чудовим прикладом для побудови підсистеми для роботи з картою.

2 РЕАЛІЗАЦІЯ

2.1 Моделювання та аналіз програмного забезпечення

Систему було побудовано на основі клієнт-серверної архітектури, яка є домінуючою концепцією у створенні розподілених мережних застосунків і передбачає взаємодію та обмін даними між ними. Було використано різноманітні технології для досягнення бажаного результату (Рисунок 2.1).

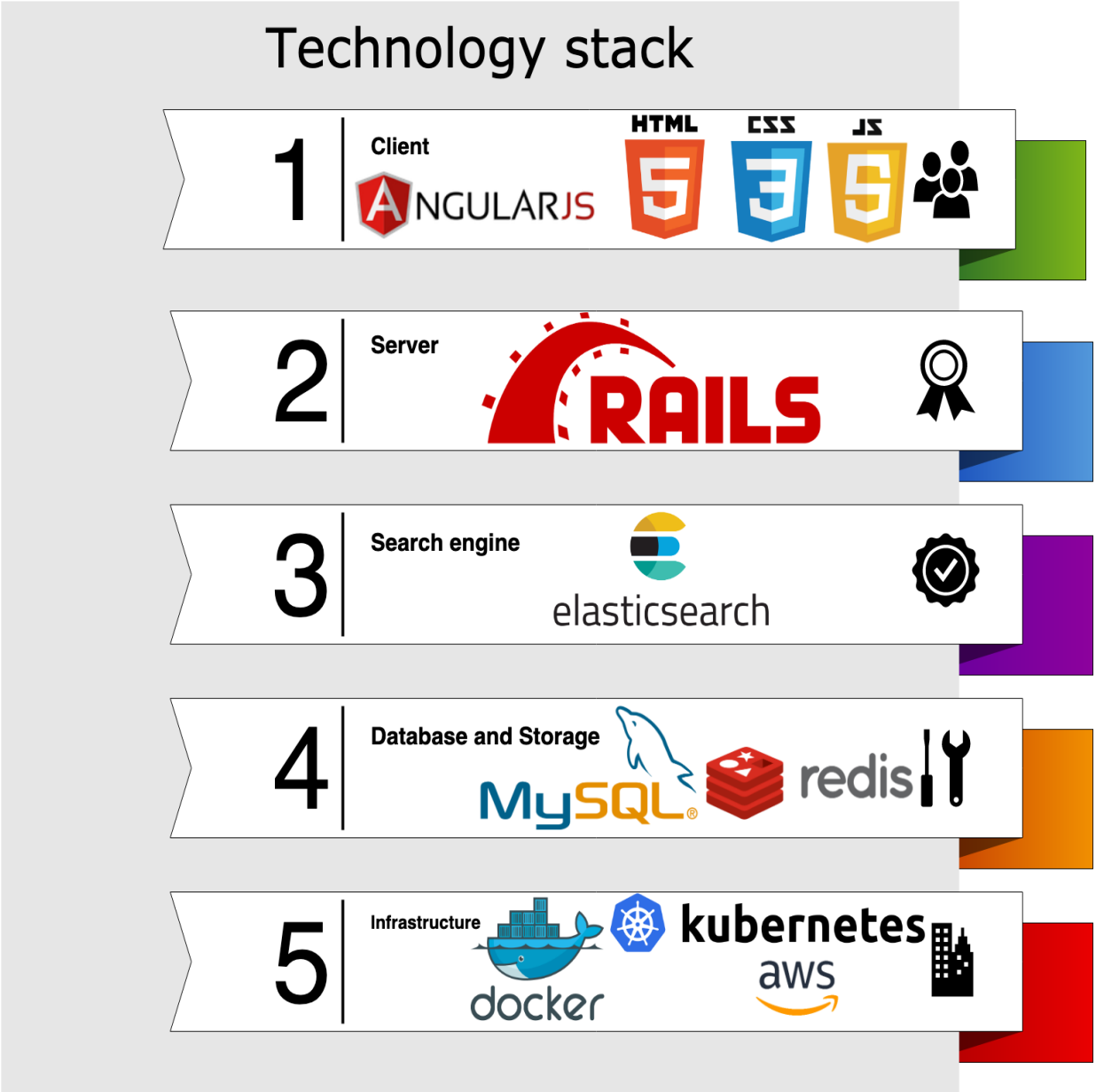

Рисунок 2.1 - Стек технологій дипломного проекту

Для серверної частини було використано мову програмування Ruby та фреймворк Ruby on Rails. На даний момент Ruby є однією з найпопулярніших у світі мов програмування. Вона також є одною з найбільш схожих на людино-подібну мову кодування, яка дуже нагадує англійську мову. Це полегшує освоєння та кодування. Її творець, Юкіхіро Мацумото, розробив мову, що має на меті бути «справжньою об'єктно-орієнтованою мовою». Випущена в 1995 році, стала улюбленою мовою з незліченими розробниками Ruby, що будують нові проекти на основі його найбільш популярного фреймворку Ruby on Rails.

Ruby on Rails або Rails – це фреймфорк для розробки веб-додатків, написаний на Ruby під ліцензією MIT. Rails – це модель-інтерфейс-контролер (MVC) фреймворк, що забезпечує за замовчуванням підтримку бази даних, веб-служби та веб-сторінок.

Для управління реляційною базою даних було вибрано MySQL. MySQL є відкритою системою управління реляційними базами даних. Вона заснований на мові запитів структури (SQL), який використовується для додавання, видалення та модифікації інформації в базі даних. Стандартні команди SQL, такі як ADD, DROP, INSERT і UPDATE, можна використовувати з MySQL. MySQL спроектований на базі моделі клієнт-сервер. Основним компонентом MySQL є сервер MySQL, який обробляє всі команди бази даних. Сервер MySQL доступний як окрема програма для використання в мережевому середовищі клієнт-сервер і як бібліотека, яка може бути вбудована в окремі програми.

Кожен веб-додаток потребує механізм черги для обробки фонових завдань. Фонові завдання – це просто завдання, які виконуються у фоновому режимі, поза межами основного циклу веб-відповіді.

Було використано фреймворк ActiveJob та адаптер до нього Sidekiq для максимально швидкого реагування веб-сторінки, що означає отримання даних на екрані, виконання запиту та повернення контролю користувачеві. Фонові завдання створюються для обробки завдань, які вимагають часу для завершення або не є критичними для відображення результатів на екрані.

ActiveJob – це фреймворк для оголошення завдань і залучення їх до роботи на різних серверах. Ці робочі завдання можуть бути різною сутністю, починаючи від регулярних запланованих очищень, до виставлення рахунків та розсилок. Все, що може бути поділене на невеликі одиниці роботи і працювати паралельно.

Sidekiq – повнофункціональний адаптер для обробки фонових задач. Він прагне бути простим для інтеграції з будь-яким сучасним додатком Rails і є більш продуктивний, ніж інші існуючі рішення. Sidekiq використовує Redis для зберігання всіх своїх завдань і робочих даних.

Redis – сховище даних в пам'яті, що використовується як база даних, кеш і брокер повідомлень. Він підтримує такі структури даних, як рядки, хеші, списки, набори, відсортовані набори з діапазонними запитами, растровими зображеннями, геопросторовими індексами з радіусами. Redis має вбудовану реплікацію, транзакції та різні рівні збереження на диску.

Кожен веб-додаток з великою кількістю даних потребує швидкий механізм пошуку та обробки інформації. Для вирішення цієї проблеми було використано Elasticsearch. Elasticsearch – це пошуковий двигун з відкритим вихідним кодом, RESTful, побудований на Apache Lucene. З моменту свого випуску в 2010 році Elasticsearch швидко став найпопулярнішою пошуковою системою, і зазвичай використовується для аналізу журналів, повнотекстового пошуку, розвідки безпеки, бізнес-аналітики та випадків використання оперативної розвідки.

Клієнт був створений за допомогою Html, Css, JavaScript та фреймворку AngularJS.

AngularJS – це структурна основа для динамічних клієнтських веб-додатків. Технологія дозволяє використовувати HTML як мову шаблону і розширювати синтаксис HTML, щоб чітко і лаконічно висловлювати компоненти програми. Прив'язка даних AngularJS та ін'єкція залежностей усувають більшу частину коду. І все це відбувається в браузері, що робить його ідеальним партнером з будь-якою серверною технологією.

Середовище розробки повністю покрите за допомогою Docker. Docker дозволяє упакувати програму або службу з усіма її залежностями в стандартизовану одиницю. Цю одиницю зазвичай позначено як зображення Docker. Включено все, що потрібно застосувати. Зображення Docker містить код, середовище виконання, системні бібліотеки та все інше.

Для розгортання веб-додатку та підтримки було використано Kubernetes.

Kubernetes – це портативна, розширювана платформа з відкритим вихідним кодом для керування робочими навантаженнями та послугами в контейнерах, що полегшує як декларативну конфігурацію, так і автоматизацію. Вона має велику, швидко зростаючу екосистему. Послуги, підтримка та інструменти Kubernetes широко доступні. Google відкрив проект Kubernetes у 2014 році. Kubernetes базується на десятирічному досвіді, який Google здійснює за допомогою масштабних виробничих навантажень у поєднанні з найкращими ідеями та практикою спільноти.

2.1.1 Проектування серверної частини та бізнес логіки

Проектування бізнес логіки було почато з розробки діаграми варіантів використання веб-додатку (Рисунок 2.2). Найпростіша діаграма варіантів використання являє собою уявлення про взаємодію користувача з системою, яка показує взаємозв'язок між користувачем і різними випадками використання, в яких задіяний користувач. Діаграма варіантів використання може ідентифікувати різні типи користувачів системи та різні випадки використання.

Система має п’ять ролей для користувачів: новий користувач, зареєстрований в системі користувач, експерт, що буде проводити ревізії, адмін сайту та користувач, що буде створювати електронну збірку інформації щодо закладів харчування. Останні три ролі для користувачів будуть мати доступ до адмін панелі, яка є недоступною частиною системи для звичайних користувачів.

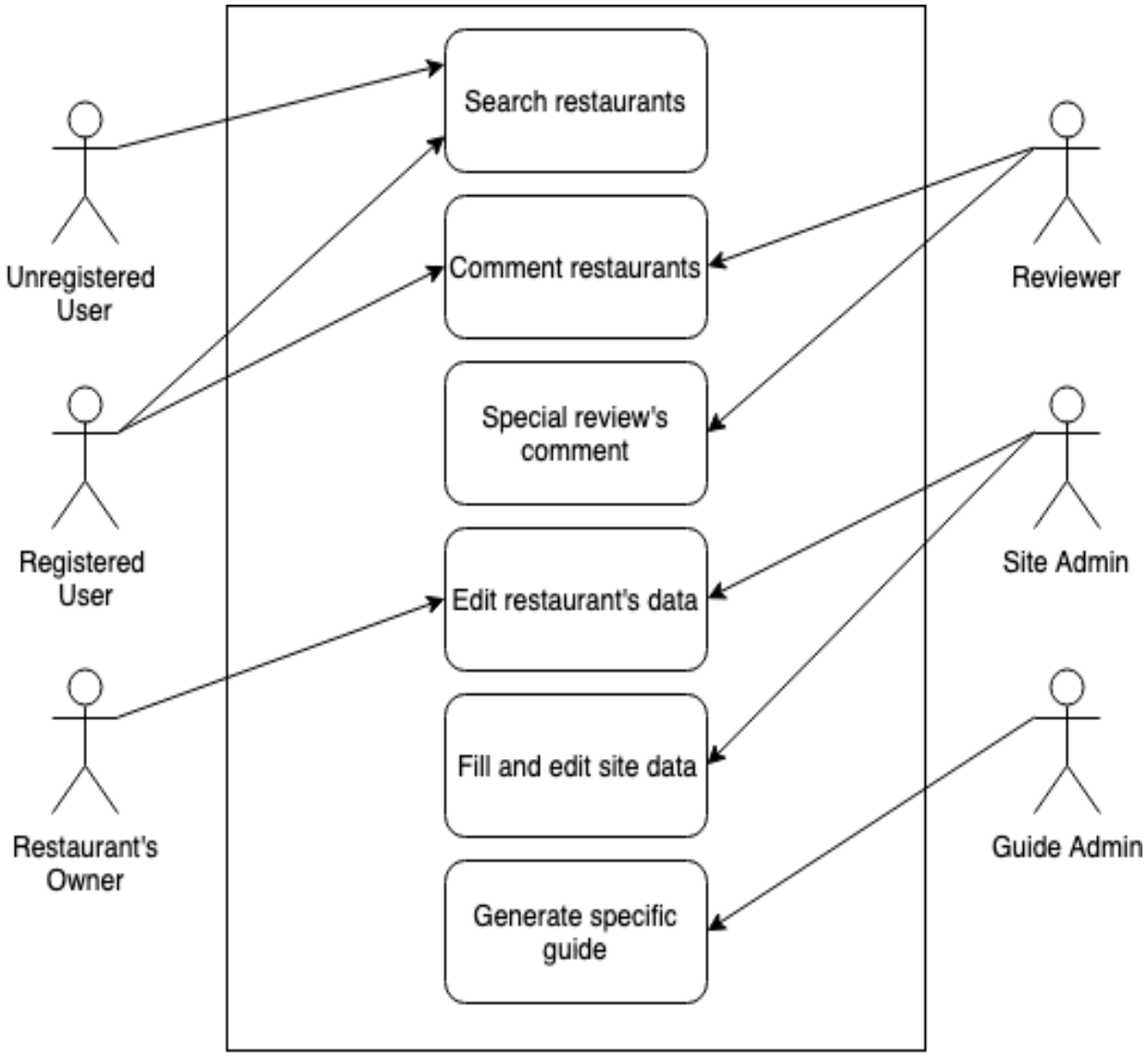

Рисунок 2.2 - Діаграма варіантів використання

Для функціювання продукту було розроблено такі моделі-класи: Site, Account, City, Establishment, EstablishmentInfo, Guide, GuideElement, GuideFilter, GuideSection, Location, OwnershipAff, SiteAff, Review, ReviewText, Comment. Більш детальна інформація знаходиться в таблиці опису класів системи (Таблиця 2.1).

Таблиця 2.1 – Опис класів системи

Клас **(**структура**)**	Опис
Site	Клас, який містить загальну бізнес-логіку системи та локалізацію кожного веб-додатку
City	Клас, який містить бізнес-логіку розташування закладів харчування.
Location	Клас, який містить бізнес-логіку для повної деталізації та зв’язки компонентів Establishment та City.
Account	Клас, який містить бізнес-логіку для створення та манікулювання користувачами. Включає в собі криптування паролів, валідацію полів користувача та інше.
Establishment	Клас, який містить бізнес-логіку компоненту закладу харчування. Включає в собі модуль з вирахування поточного рейтингу закладу харчування.
EstablishmentInfo	Клас, який містить бізнес-логіку для повної деталізації компоненту Establishment.
Review	Клас, який містить усі дані щодо ревізії. Також присутній механізм стану для підсумування пророблено роботи та прогресу.

Продовження таблиці 2.1 - Опис класів системи

ReviewText	Клас, який містить бізнес-логіку для повної деталізації компоненту Review з можливістю мовної локалізації
Comment	Клас, який містить поточні дані залишені користувачем, щодо закладу харчування.
OwnershipAff	Клас, який містить бізнес-логіку для повної деталізації та зв’язки компонентів Establishment та Account
SiteAff	Клас, який містить бізнес-логіку для повної деталізації та зв’язки компонентів Site та Account
Guide	Клас, який містить бізнес-логіку для повної деталізації електронної збірки інформації щодо закладів харчування.
GuideFilter	Клас, який містить бізнес-логіку для повної деталізації параметрів за якими відбудеться генерація Guide
GuideElement	Клас, який містить бізнес-логіку для створення та манікулювання логічної одиниці Guide
GuideSection	Клас, який містить бізнес-логіку для створення та манікулювання логічної секції GuideElement

Після створення одномовної повноційної системи, постала проблема інтернаціоналізації. Інтернаціоналізація є складною проблемою для кожного веб-додатку. Природні мови розрізняються багатьма способами, тому важко забезпечити інструменти для вирішення всіх проблем одночасно. З цієї причини було використано модуль I18n, який засереджується на:

- надання та підтримки англійської та подібних мов
- полегшення налаштування та розширення всіх інших мов

Перш ніж виконувати переклади, потрібно вирішити, які мови будуть підтримуватися. Було вирішено підтримувати українську та англійську, причому остання встановлюється за замовчуванням. Відображано це у конфураційному файлі ‘config/application.rb’ (Рисунок 2.3).

```
# The default locale is :en and all translations from config/locales/*.rb,yml are auto loaded.
# config.i18n.load_path += Dir[Rails.root.join('my', 'locales', '*.{rb,yml}').to_s]
config.i18n.available_locales = [:en, :ua]
config.i18n.default_locale = :en
config.i18n.fallbacks = [:en]
```

Рисунок 2.3 - Конфігурація I18n

Після конфігурації було вибрано метод зберігання перекладених слів в окремому файлі, а також зробити правильну версію слова на основі вибраної мови. За замовчуванням Rails використовує файли YAML, які повинні зберігатися в каталозі ‘config/locales’. Переклади для різних мов зберігаються в окремих файлах, і кожен файл має назву мови, що він перекладає.

Для функціювання продукту було розроблено такі контролери-класи: FrontController, SessionsController, RedirectionController, AccountsController, EstablishmentsController, LocationsController, ApplicationsController, MainController, AdminController, MailingController, ApiController. Кожен

контролем має певну кількість методів, що виконують обробку запитів (Рисунок 2.4).

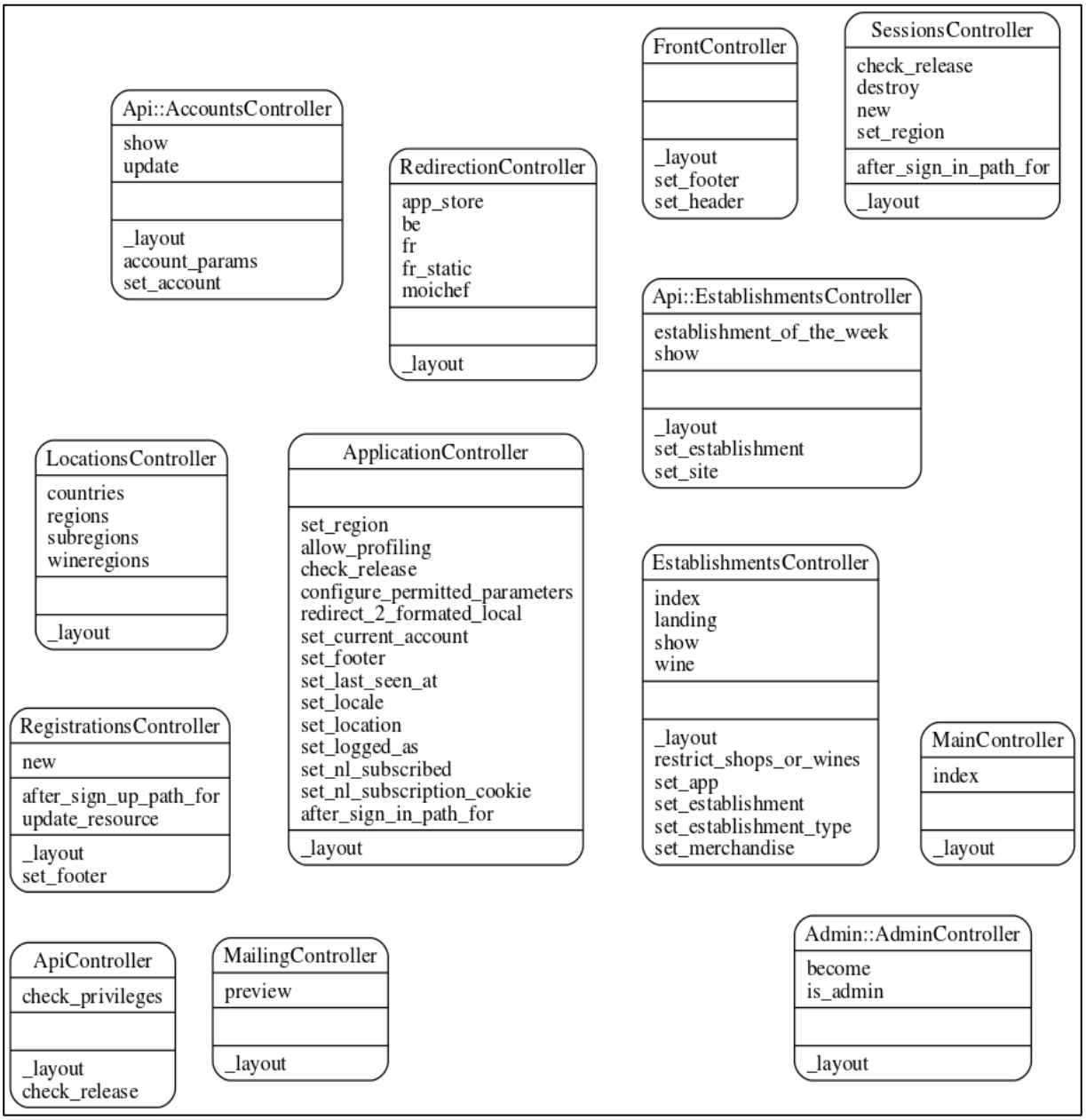

Рисунок 2.4 - Схема контролерів

Було спроектовано та розроблено такі підчастини системи:

1) Deep Search: оскільки об’єм даних дуже великий, постала проблема швидкої обробки та фільтрування даних. Кожного разу робити запит в базу даних було не доцільно і дуже повільно. Функція автодоповнення випадаючого меню повинна спрацьовувати моментально. Тому було

прийнято рішення використати Elasticsearch. Elasticsearch - це відкрита, широко розповсюджувана, легко масштабована пошукова система, розроблена на базі Lucene. Надає розподілений, мультиарендний повнотекстовий пошуковий рушій з HTTP веб-інтерфейсом і підтримкою безсхемних JSON документів. Elasticsearch має можливість розподілення, індекси можуть бути розділені по шардах, при чому кожен шард може мати нуль чи більше реплік. Кожен вузол містить один чи більше шардів і діє як координатор делегування операцій на потрібний шард. Балансування та маршрутизація виконується автоматично. Для цієї інтеграції було використано модуль Elasticsearch::Model. Він спрямований на спрощення інтеграції класів Ruby або моделей у програмах Ruby on Rails з пошуковим і аналітичним механізмом Elasticsearch. Було створено клас EstablishmentIndexes, який відповідає за конфігуляцію на налаштування індексів для Elasticsearch. Індекс буде включати в собі інформацію про кожного індексованого закладу харчування. Було проіндексовано такі поля класу Establishment: 'name', 'type', 'chef', 'zip_code', 'id', 'state', 'review_text', 'city', 'notes', 'category', 'moment', 'service'. Для пошуку не тільки за індексованими полями було створено фасети: [:toque, :region, :country, :country_code, :zip_code, :mark, :city, :subregion, :name, :chef, :state, :type, :review_vintage, :review_state, :admin_review_state, :icon_price, :category, :site_category, :shop_category, :cuisine, :booking, :moment, :service, :last_inquiry_date, :has_coordinates, :bookable, :last_reviewer_nickname, :users_rating, :pop, :timetable_flattened, :timetable_test, :parameterized_city, :has_photos, :accountant, :host]. Кожний фасет описаний відповідним методом з більш детальною логікою з прив’язкою до полів класу Elasticsearch.

2) Map: для динамічної карти було вибрано Leaflet. Це бібліотека з відкритим вихідним кодом, написана на JavaScript, призначена для

відображення карт на веб-сайті. Leaflet дозволяє розробкику, не знайомому з геоінформаційною системою, легко відображати растрові карти, що складаються з маленьких фрагментів - тайлів, додаткових шарів, накладок які накладаються на основний шар тайлу. Фрагменти можуть бути інтерактивними, наприклад, показувати підсказку при кліку по маркеру. В якій знаходиться більш детальна інформація про заклад харчування. Карта була інтегрована в AngularJS контроллер, також через нього відбувалися всі операції та дії користувача. Дані було заіндексовано в Elasticsearch, тому це гарантує швидкодію та бажаний результат (Рисунок 2.5).

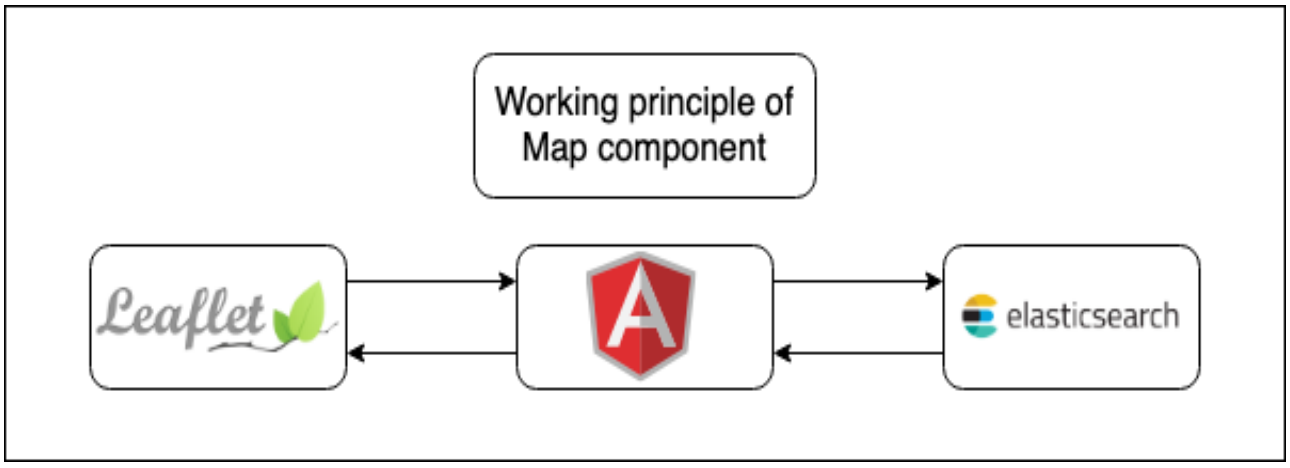

Рисунок 2.5 - Принцип роботи компоненту Map

3) Admin panel and Review system: Частина веб-сайту яка буде доступна тільки для окремих ролей системи. Було спроектовано і розроблено Admin::API інтерфейс для обробки запитів цієї підсистеми. Admin::API інтерфейс спроектований на основі REST архітектури. Було розроблено зручні форми для заповнення необхідної інформації після ревізії того чи іншого закладу харчування. Оцінка якості закладу харчування складається з опису та числового позначення. Числовий показник може знаходитись в діапазоні від 1 до 20 з інтервалом 0,5. Загальний показник якості вимірюється в умовних одиницях -

кухарський капелюх. Їх кількість залежить від числового показнику. Клас Review тримає всю логіку цього процесу. Алгоритм визначення кількості кухарських капелюхів доволі простий і може кастомізуватися в залежності від локалізації веб-додатку. Наприклад, за замовчуванням було закодовану таку логіку (Рисунок 2.6). Так маючи числовий показник 9,5, будемо мати загальний показник якості, який рівний одиниці, тобто одному капелюху. Маючи числовий показник 12,5, будемо мати загальний показник якості 3, тобто три капелюха. Числовий показник і загальний показник (капелюх) відображено на клієнтській частині разом.

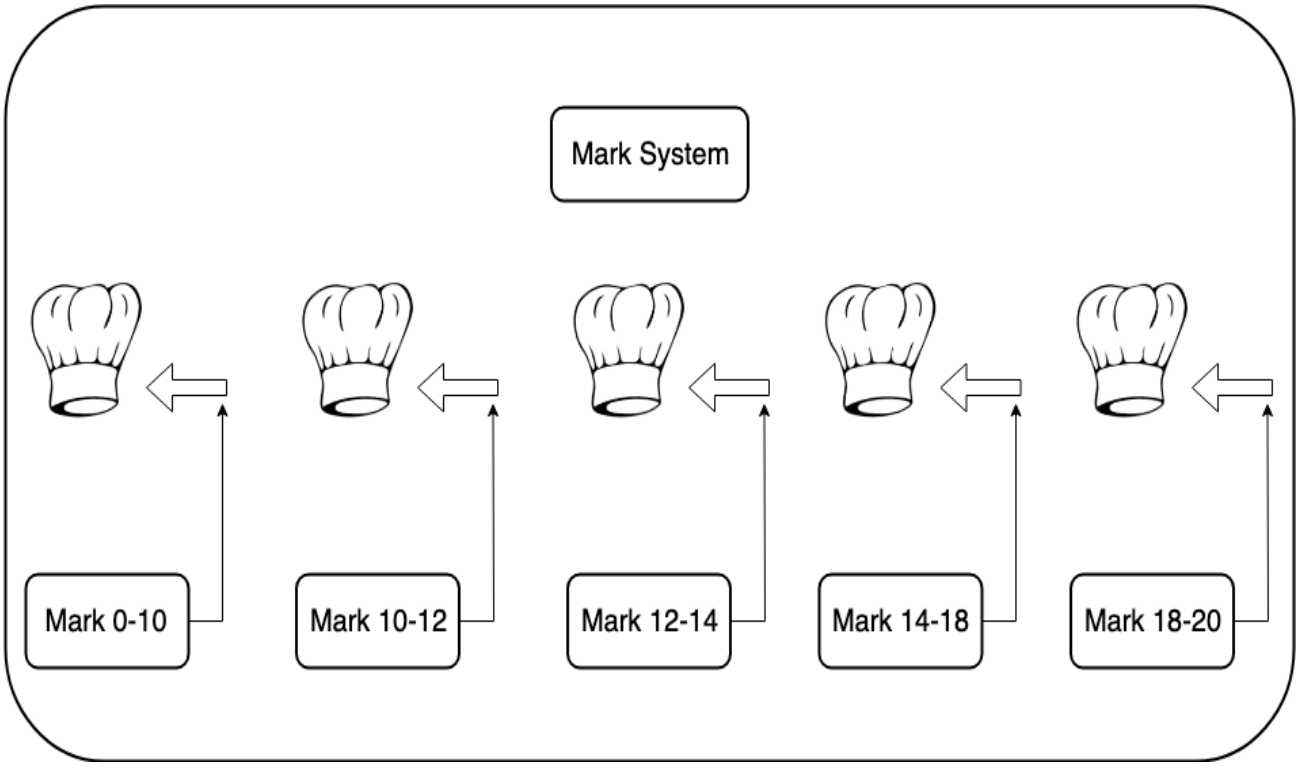

Рисунок 2.6 - Показник якості в умовних одиницях

4) Guide generation: Система для генерації та об’єднання даних за відповідними фільтрами, які можна буде в будь-який час експортувати в xml-файл. Згенерований файл буде доступний тільки для відповідного адміна. Для зберігання інформації про заклади

харчування, яка буде готова бути експортована в xml формат, було використав структуру данних – дерево, для поетапного відображення даних на клієнті. В такому вигляді зберігється Guide в базі данних (Рисунок 2.7). Для створення та експорту Guide в xml формат використовуються фонові задачі в Ruby on Rails – Active Job, так як процес може зайняти доволі довгий термін. Active Job - це фреймворк для оголошення фонових завдань і їх запускання по різних бекендах для черг. Ці задачі можуть бути всім, що може бути вилучено в невеликі робочі частини і запускатися паралельно (Рисунок 2.8).

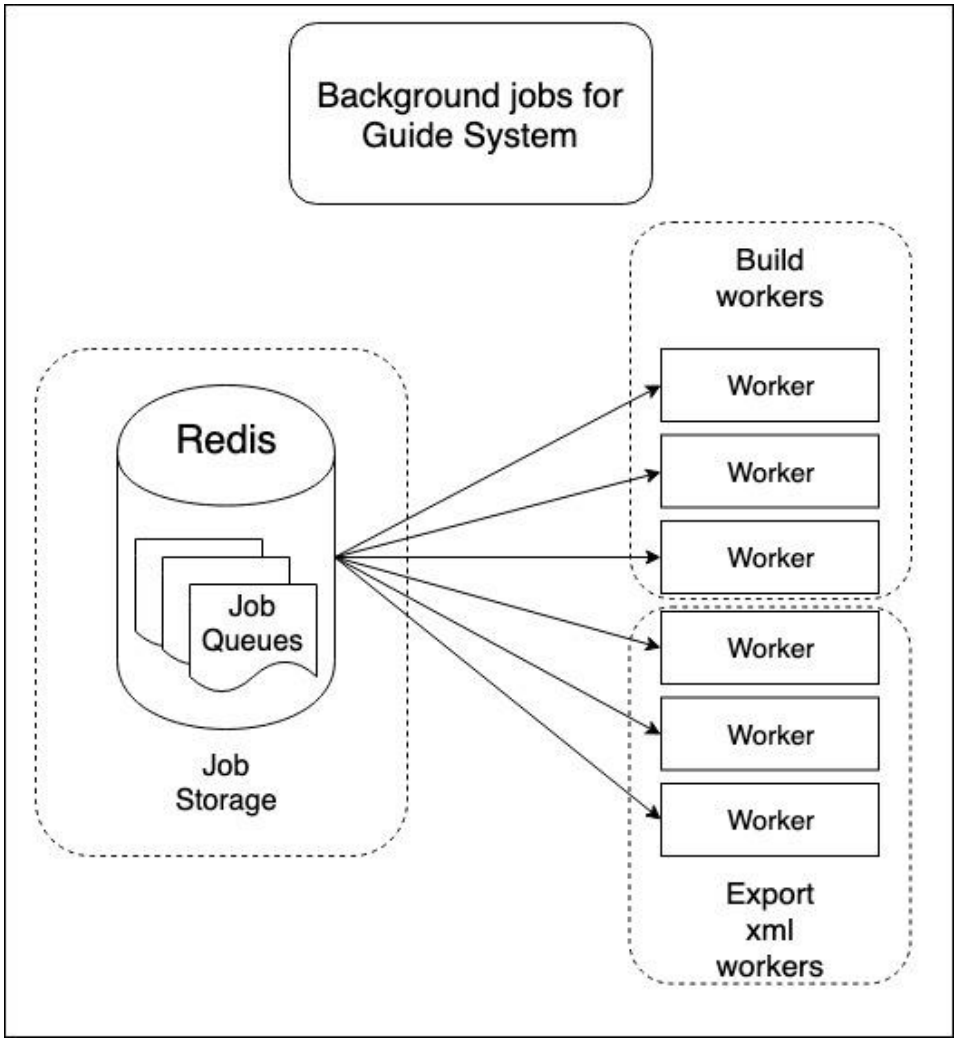

Рисунок 2.8 - Структура фонових задач

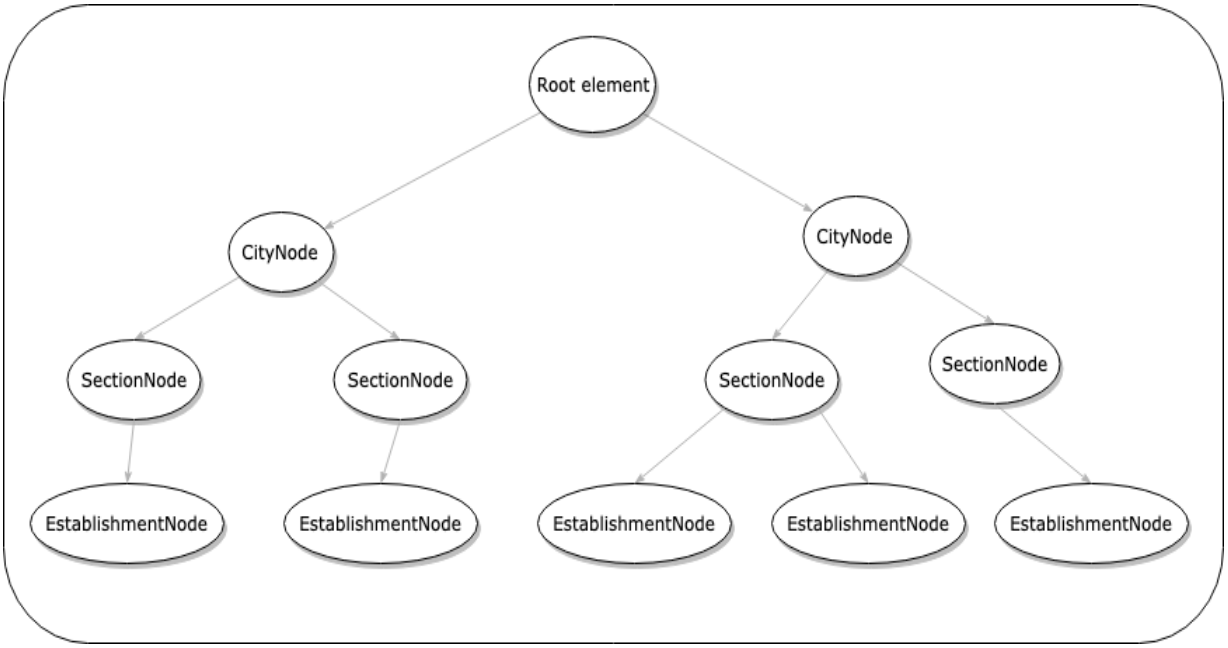

Рисунок 2.7 - Структура Guide

2.1.2 Проектування бази данних

Спроектована MySQL база даних має відповідні таблиці:

1) sites;
2) accounts;
3) cities;
4) locations;
5) ownership_affs;
6) site_affs;
7) establishments;
8) establishment_infos;
9) reviews;
10) review_texts;
11) comments;
12)guides;
13) guide_elements;
14) guide_filters;

15) guide_section;

Назви таблиць було підібрано відповідно до бізнес моделі та конвенції імен в Rails.

Для збереження даних про модель-клас Site призначена таблиця "sites" (Рисунок 2.9).

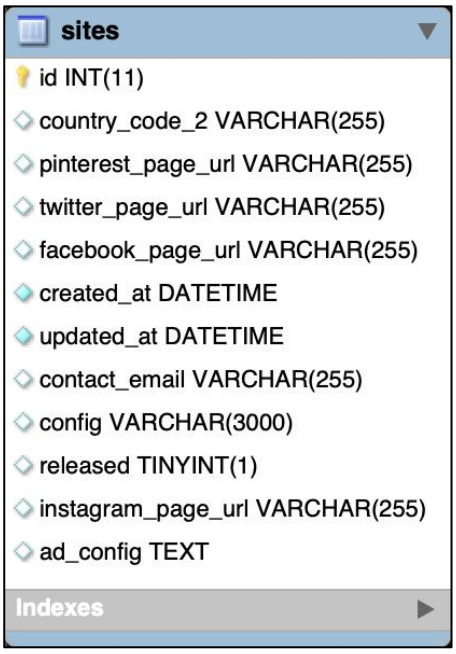

Рисунок 2.9 - Структура таблиці “sites”

Для збереження даних про модель-клас OwnershipAff призначена таблиця " ownership_affs" (Рисунок 2.10).

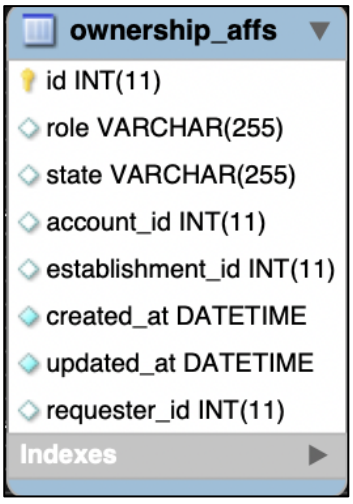

Рисунок 2.10 - Структура таблиці “ownership_affs”

Для збереження даних про модель-клас Account призначена таблиця "accounts" (Рисунок 2.11).

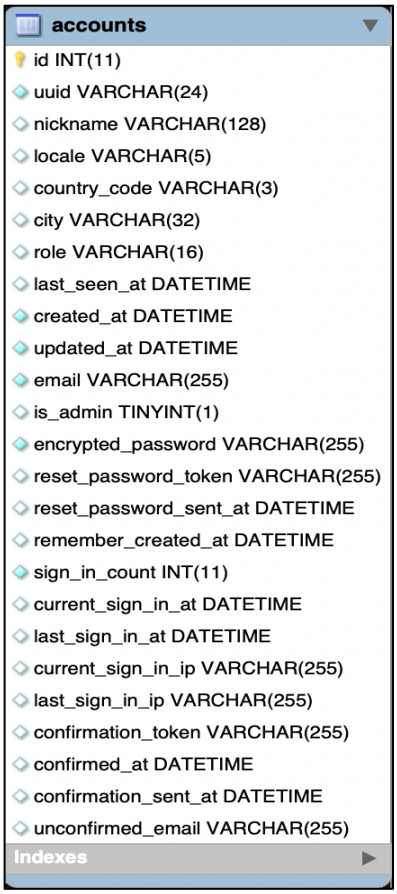

Рисунок 2.11 - Структура таблиці “accounts”

Для збереження даних про модель-клас SiteAff призначена таблиця "site_affillations" (Рисунок 2.12).

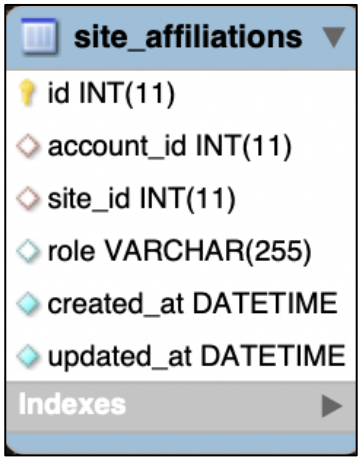

Рисунок 2.12 - Структура таблиці “site_affilations”

Для збереження даних про модель-клас City призначена таблиця "cities" (Рисунок 2.13).

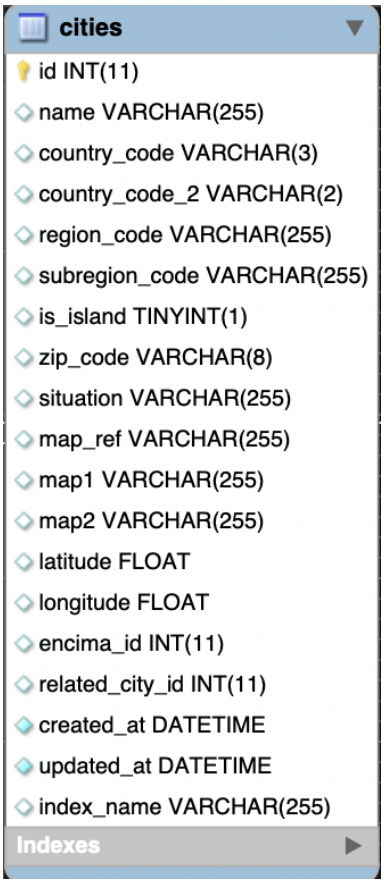

Рисунок 2.13 - Структура таблиці “cities”

Для збереження даних про модель-клас OwnershipAff призначена таблиця " ownership_affs" (Рисунок 2.14).

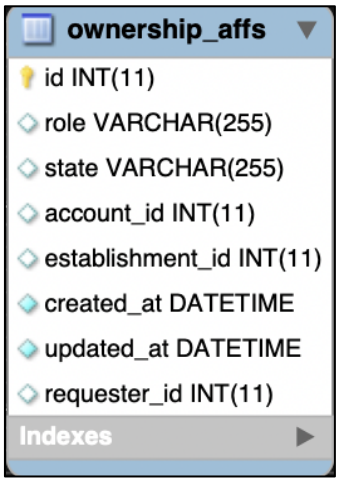

Рисунок 2.14 - Структура таблиці “ownership_affs”

Для збереження даних про модель-клас Location призначена таблиця "locations" (Рисунок 2.15).

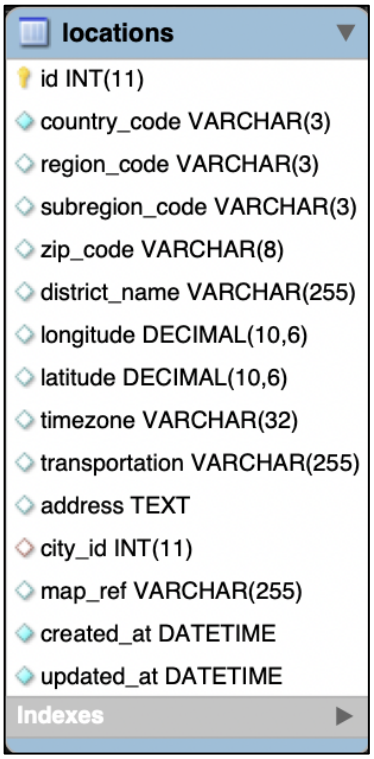

Рисунок 2.15 - Структура таблиці “locations”

Для збереження даних про модель-клас EstablishmentInfo призначена таблиця "establishment_infos" (Рисунок 2.16).

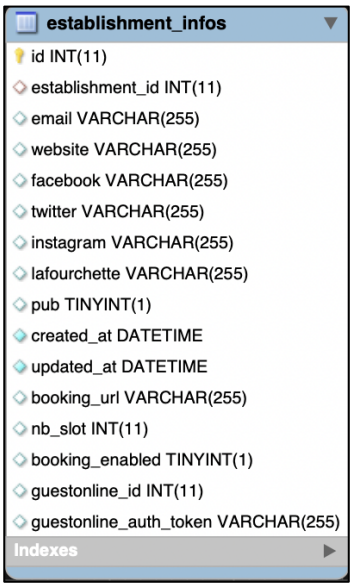

Рисунок 2.16 - Структура таблиці “establishment_infos”

Для збереження даних про модель-клас Establishment призначена таблиця "establishments" (Рисунок 2.17).

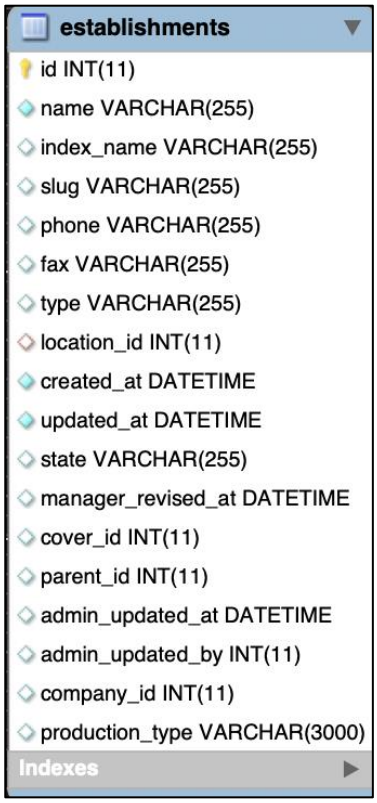

Рисунок 2.17 - Структура таблиці “establishments”

Для збереження даних про модель-клас Comment призначена таблиця "comments" (Рисунок 2.18).

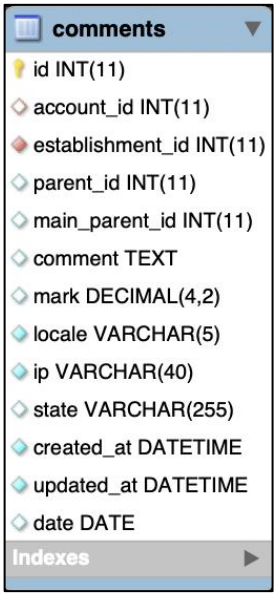

Рисунок 2.18 - Структура таблиці “comments”

Для збереження даних про модель-клас Review призначена таблиця "reviews" (Рисунок 2.19).

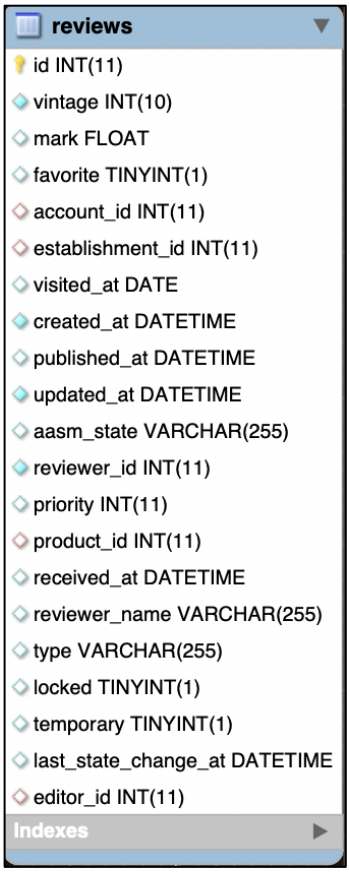

Рисунок 2.19 - Структура таблиці “reviews”

Для збереження даних про модель-клас ReviewText призначена таблиця "review_texts" (Рисунок 2.20).

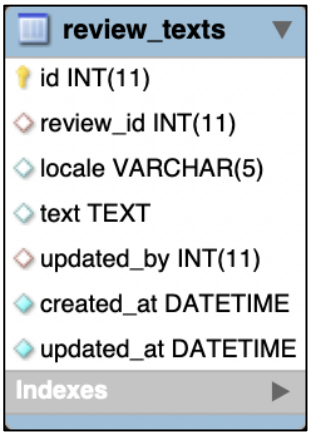

Рисунок 2.20 - Структура таблиці “review_texts”

Для збереження даних про модель-клас Guide призначена таблиця "guides" (Рисунок 2.21).

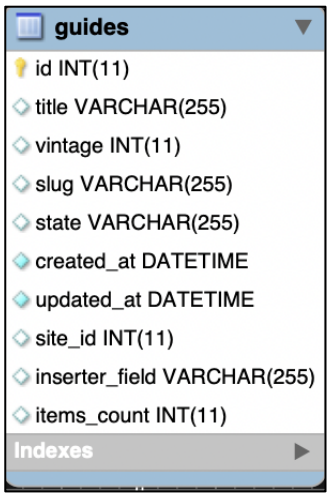

Рисунок 2.21 - Структура таблиці “guides”

Для збереження даних про модель-клас GuideElement призначена таблиця "guide_elements" (Рисунок 2.22).

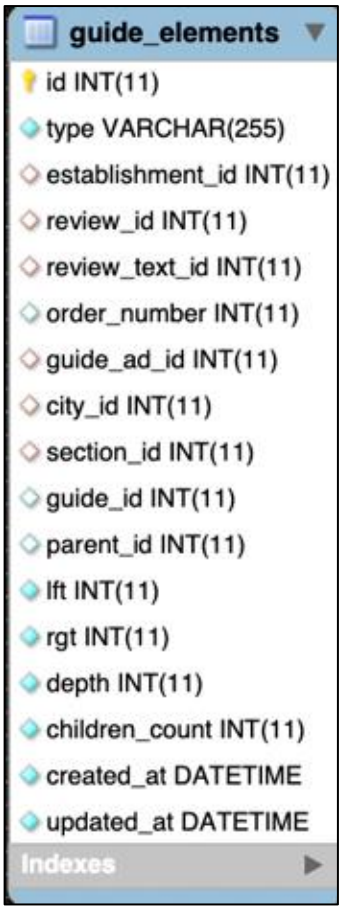

Рисунок 2.22 - Структура таблиці “guide_elements”

Для збереження даних про модель-клас GuideSection призначена таблиця "guide_sections" (Рисунок 2.23).

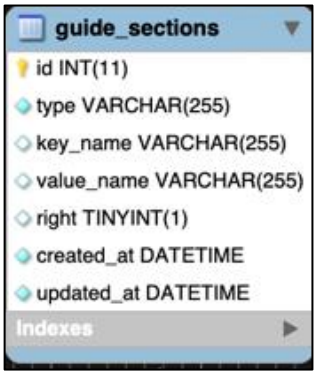

Рисунок 2.23 - Структура таблиці “guide_sections”

Для збереження даних про модель-клас GuideFilter призначена таблиця "guide_filters" (Рисунок 2.24).

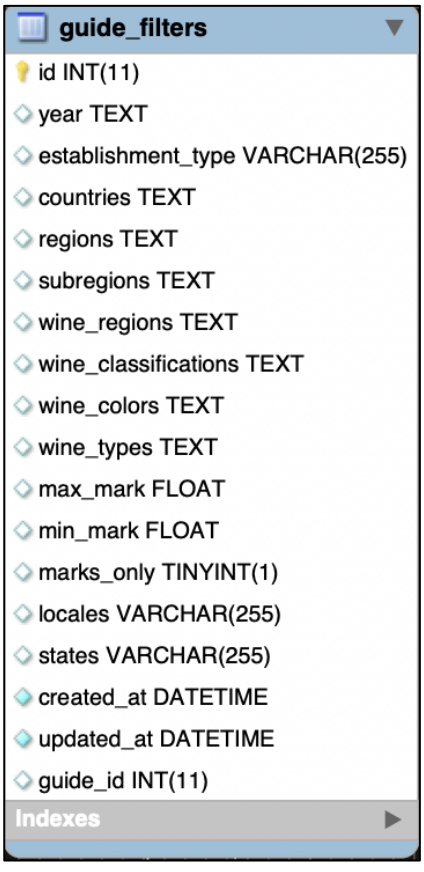

Рисунок 2.24 - Структура таблиці “guide_filters”

2.2 Функціонал клієнтської частини

Кожна спроектована підчастина системи має відповідний клієнтський інтерфейс. Головна сторінка веб-додатку має випадаюче меню для фільтації та пошуку закладів харчування [Рисунок 2.25].

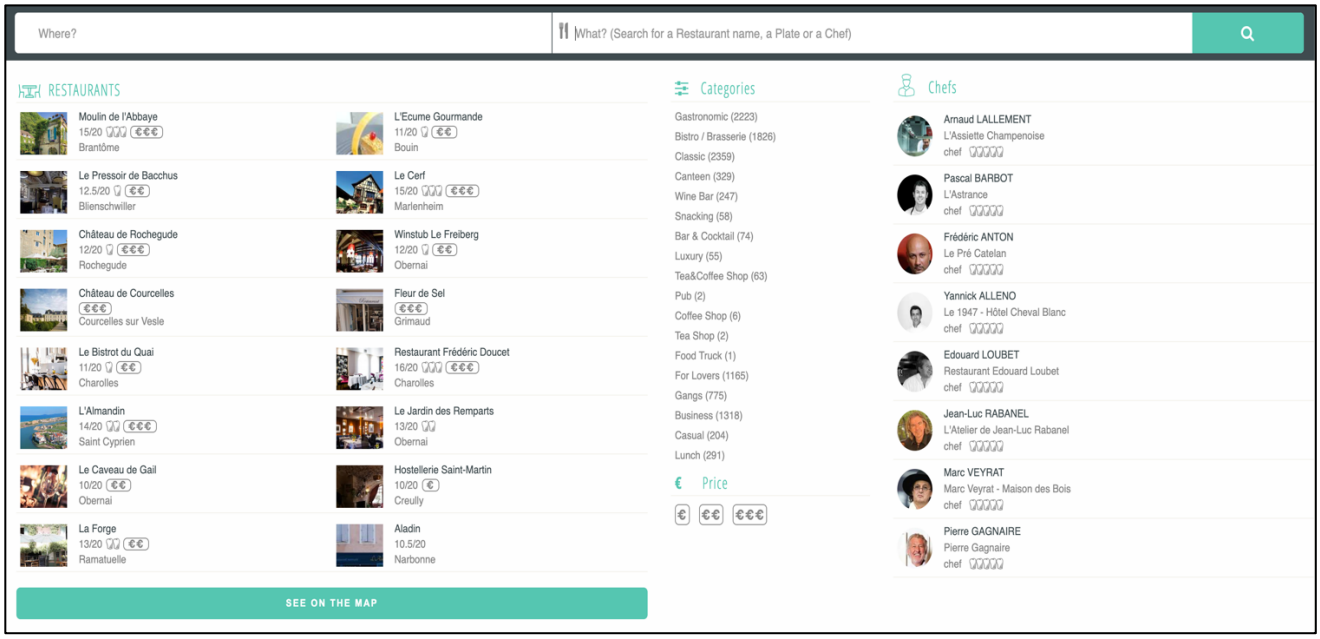

Рисунок 2.25 - Випадаюче меню пошуку

Користувачі мають доступ до карти з маркерами. Також кожен користувач зможе знайти повну інформацію про той чи інший заклад харчування та його місце розташування (Рисунок 2.26). Карта є динамічною тобто користувач зможе відсортовувати результат пошуку за заданими параметрами та за місцем розташуванням (Рисунок 2.27).

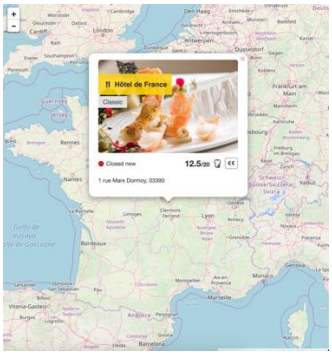

Рисунок 2.26 - Маркер закладу харчування на карті

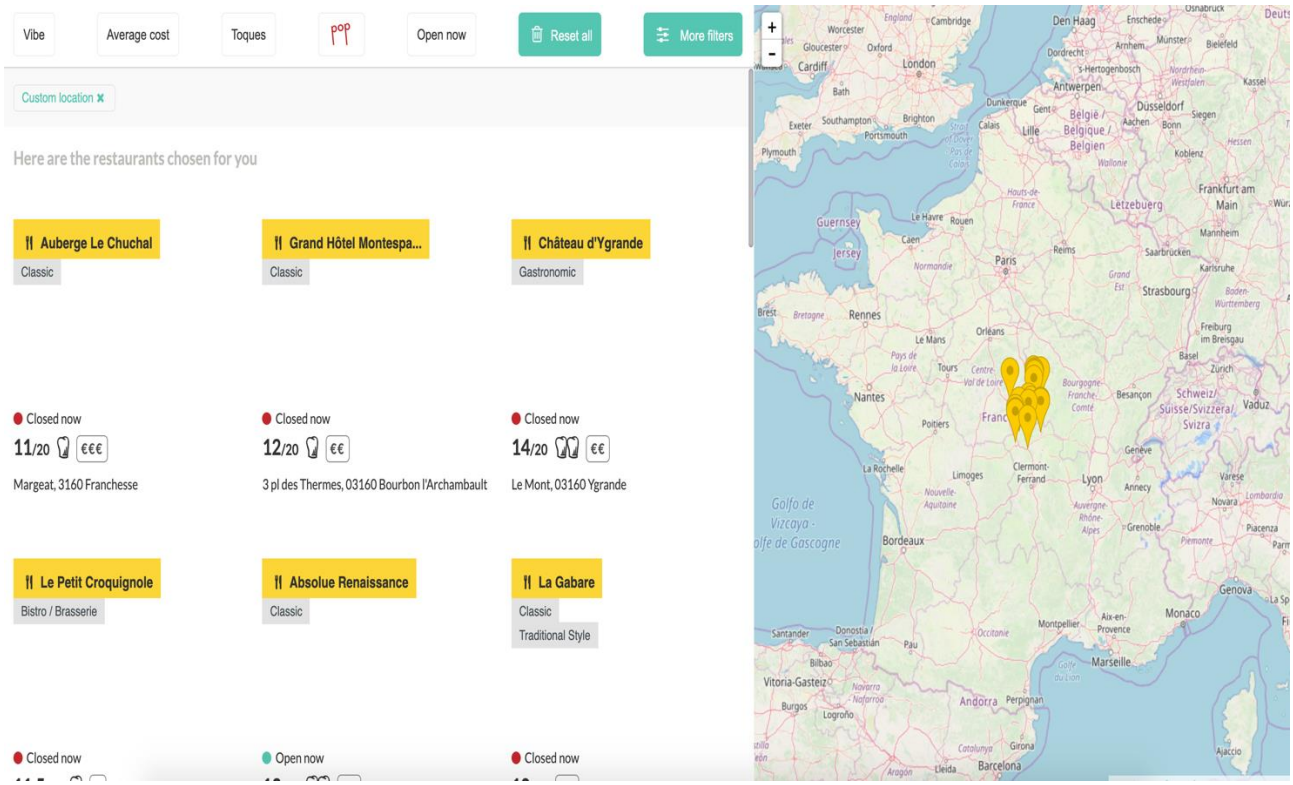

Рисунок 2.27 - Карта

Адмін панель для керування контентом доступна тільки для користувачів з відповідними ролями (Рисунок 2.28).

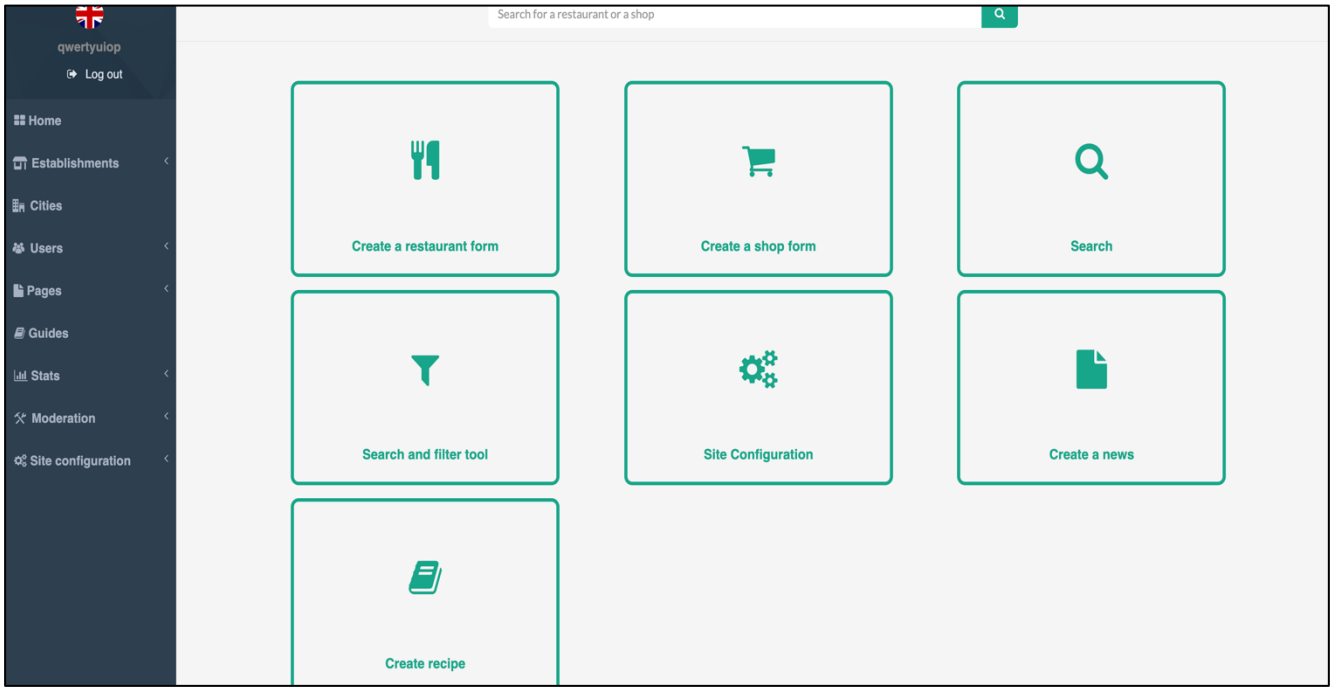

Рисунок 2.28 - Адмін панель

Також між ролями розподілено їхні права на доступ та зміну інформації. Наприклад, експерт, що буде проводити ревізії, може залишати свій звіт та ставити оцінки ресторану і не більше (Рисунок 2.29).

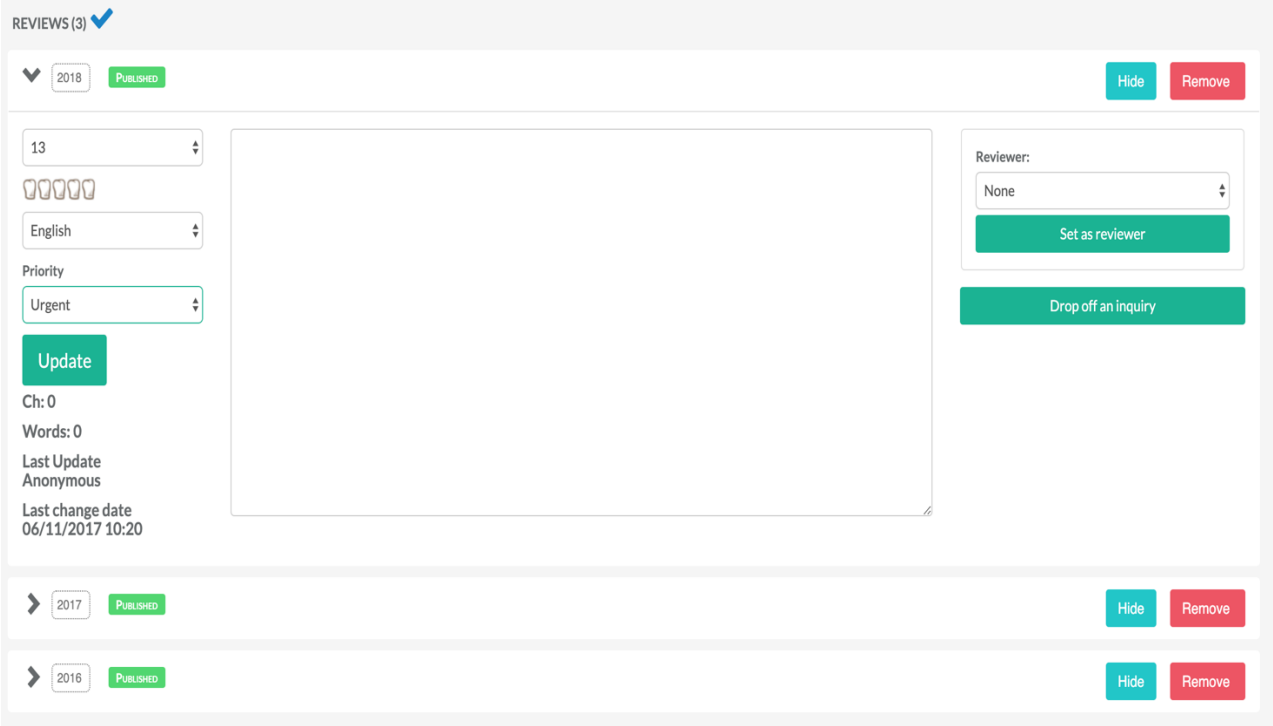

Рисунок 2.29 - Форма для заповнення звіт після ревізії

Користувач, що буде створювати електронну збірку інформації щодо закладів харчування має доступ тільки до цієї частини адмін панелі (Рисунок 2.30).

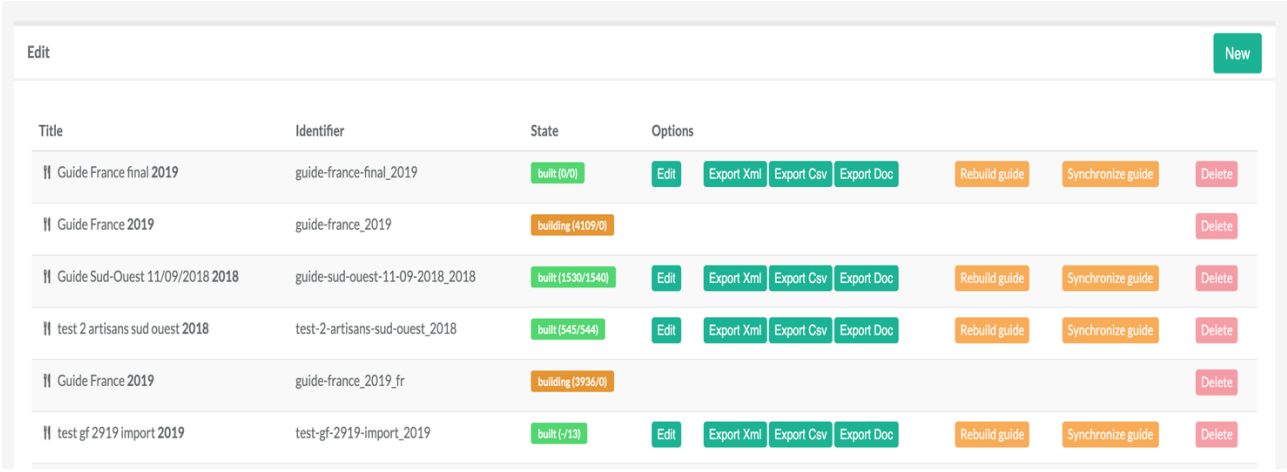

Рисунок 2.30 - Guide інтерфейс

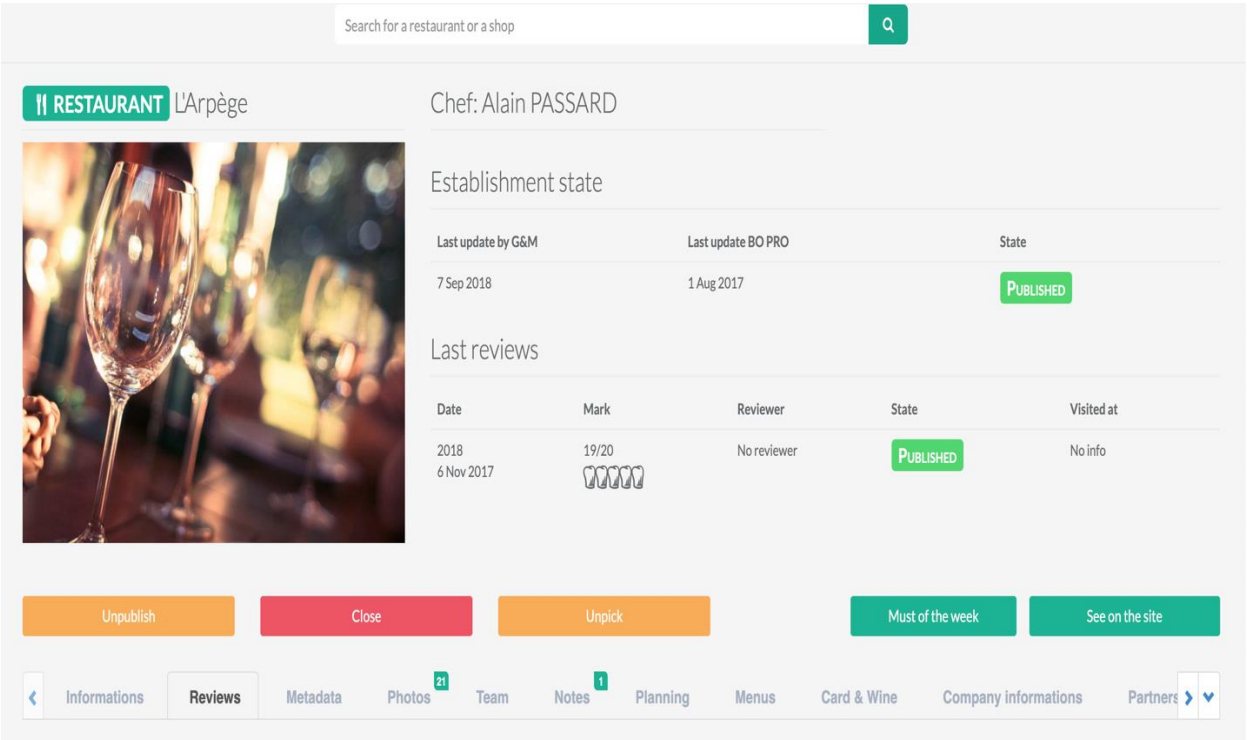

Рисунок 2.31 - Форма редагування закладу харчування

Адмін сайту та користувач, що є власником закладу харчування може змінювати та редагувати інформацію закладів харчування (Рисунок 2.31).

Щоб оцінка якості закладів харчування не була доволі суб'єктивною та не базувалася на думці одного експерта, система має другий критерій оцінки якості закладів харчування: оцінка на основі коментарів та рейтингу користувачів системи (Рисунок 2.32).

2.4 Висновки

Досліджуючи існуючі рішення систем відповідно до завдання дипломного проекту, було спроектовано систему оцінки якості закладів харчування. Було вирішено проблеми пов'язані з оцінкою та проектування системи відгуків для закладів харчування, досліджено методи зберігання та використання великих об'ємів даних для швидкого та комфортного перегляду користувачем. Виконано

завдання на розробку чіткої структури шкали оцінки, зручного інтерфейсу web-додатку та впроваджено гнучку систему коментарів. Було досліджено декілька важливих архітектурних підходів і вибрано найкращий, який підходить до завдання дипломного проекту

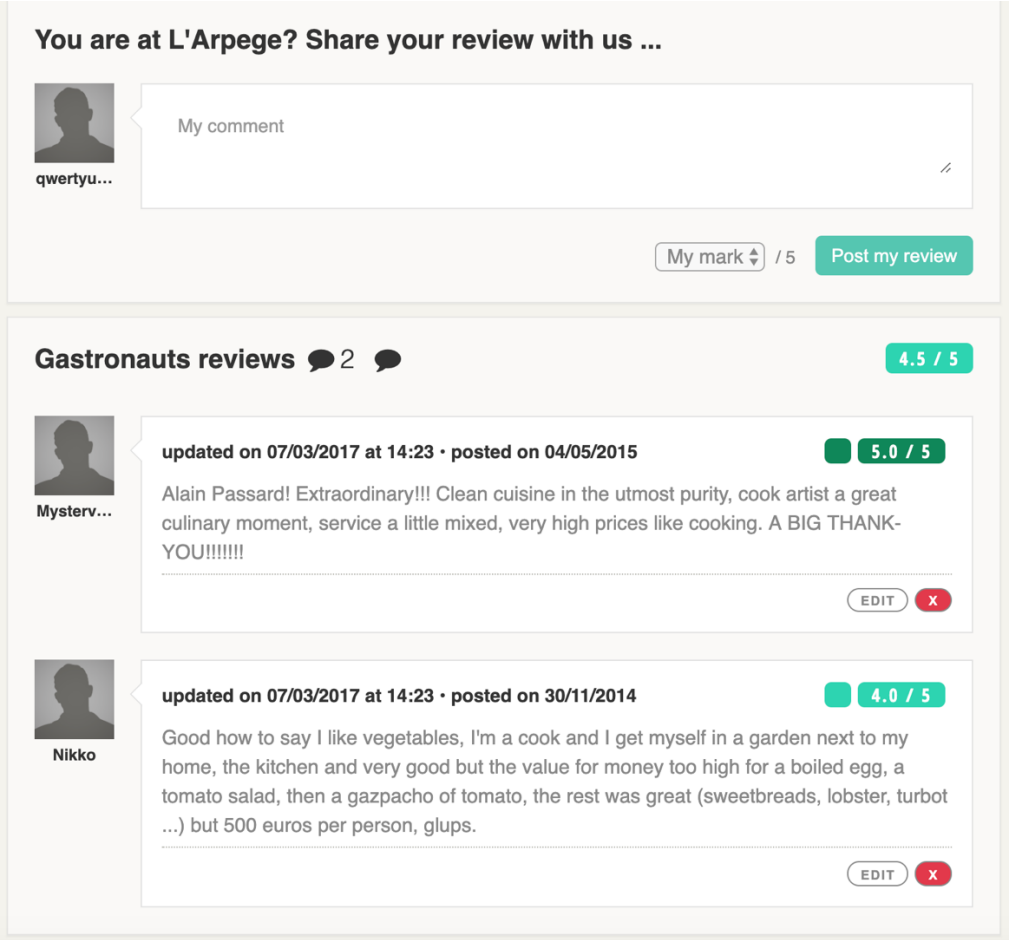

Рисунок 2.32 - Секція для коментарів

3 АНАЛІЗ ЯКОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ТА ТЕСТУВАННЯ

3.1 Аналіз якості програмного забеспечення

Тестування програмного забезпечення - це процес, що дозволяє оцінити функціональність програмного додатка з метою з'ясувати, чи відповідає розроблене програмне забезпечення зазначеним вимогам чи ні, та виявити дефекти, щоб гарантувати, що продукт не містить дефектів.

Існує три основні типи тестів, які було використано для тестування функціональності веб-додатку Rails дипломного проекту: тест моделі-класу, тести контролерів і тести інтерфейсів. Крім того, було використовано тести функцій. Ці тести функціонуватимуть як кінцева пробна перевірка готового проекту. Кожен тип тесту має своє місце в світі тестування і може бути не менш важливим. Для тестування було вибрано RSpec фреймфорк.

RSpec є інструментом тестування для Ruby, створений для принципу тестування поведінки додатку (behavior-driven development). Це найбільш часто використовувана бібліотека тестування для Ruby у виробничих додатках. Незважаючи на те, що він має дуже багатий і потужний DSL (домен-специфічна мова), за своєю суттю це простий інструмент, який можна почати використовувати досить швидко. Сподіваємося, цей посібник допоможе вам розпочати роботу, якщо ви не маєте попереднього досвіду роботи з RSpec та навіть тестування.

Для тестування моделей-класів було спроектовано такі тестові моделі: SiteTest, AccountTest, CityTest, EstablishmentTest, EstablishmentInfoTest, GuideTest, GuideElementTest, GuideFilterTest, GuideSectionTest, LocationTest, OwnershipAffTest, SiteAffTest, ReviewTest, ReviewTextTest, CommentTest. Більш детальна інформація знаходиться в таблиці опису тестових класів системи (Таблиця 3.1).

Таблиця 3.1 – Опис тестових класів системи

Клас (тестовий модуль)	Опис
SiteTest	Клас, який містить тестові модулі для класу Site. Містить тестові сценарії для валідації полів та відповідних зв’язків has_one і has_many.
CityTest	Клас, який містить тестові модулі для класу City. Містить тестові сценарії для геологічних даних та зв’язків.
LocationTest	Клас, який містить тестові модулі для класу Location. Містить тестові сценарії для валідації полів та відповідних зв’язків між Establishment та City.
AccountTest	Клас, який містить тестові модулі для класу Account. Містить тестові сценарії для валідації полів класу.
EstablishmentTest	Клас, який містить тестові модулі для класу Establishment. Містить тестові сценарії для перевірки логіки розкладу закладу та зв’язків has_one і has_many.
EstablishmentInfoTest	Клас, який містить тестові модулі для класу EstablishmentInfo. Містить тестові сценарії для валідації полів.
ReviewTest	Клас, який містить тестові модулі для класу Review. Містить тестові сценарії для перевірки логіки системи оцінювання.

Продовження таблиці 3.1 - Опис тестових класів системи

ReviewTextTest	Клас, який містить тестові модулі для класу ReviewText. Містить тестові сценарії для валідації полів класу з різними локалями.
CommentTest	Клас, який містить тестові модулі для класу Comment
OwnershipAffTest	Клас, який містить тестові модулі для класу OwnershipAff. Містить тестові сценації для відповідних зв’язків між Establishment та Account.
SiteAffTest	Клас, який містить тестові модулі для класу SiteAff. Містить тестові сценації для відповідних зв’язків між Site та Account.
GuideTest	Клас, який містить тестові модулі для класу Guide Містить тестові сценарії для створення та екпорту електронної збірки.
GuideFilterTest	Клас, який містить тестові модулі для класу GuideFilter. Містить тестові сценарії для валідації полів.
GuideElementTest	Клас, який містить тестові модулі для класу GuideElement. Містить тестові сценарії для валідації полів.
GuideSectionTest	Клас, який містить тестові модулі для класу GuideSection. Містить тестові сценарії для валідації полів.

Кожен контролер-клас має відповідний тестовий модуль з інтеграційними тестами: FrontControllerTest, MainControllerTest, LcationsControllerTest,

RegistrationsControllerTest, Admin::AdminControllerTest, MailingControllerTest, ApplicationsControllerTest, Api::EstablishmentsControllerTest, ApiControllerTest, Api::AccountsControllerTest, SesionControllerTest, EstablishmentsControllerTest, RedirectionsControllerTest. Кожний тестовий модуль має тестовий сценарій для кожного методу контролера-класа.

Було використано SimpleCov модуль для перевірки якості покриття тестами веб-додатку. SimpleCov є інструментом аналізу покриття коду для Ruby. Він використовує вбудовану в Ruby бібліотеку покриття для збору даних про покриття коду, але полегшує обробку його результатів, надаючи чистий API для фільтрації, групування, об'єднання, форматування та відображення цих результатів, надаючи вам повний пакет покриття коду, який може бути налаштовується лише на кілька рядків коду.

3.3 Висновки

Тестування має багато переваг, і одним з найважливіших є економічна ефективність. Тестування може заощадити гроші в довгостроковій перспективі. Розробка програмного забезпечення складається з багатьох етапів, і якщо помилки виявляються на ранніх стадіях, це коштує набагато менше, щоб їх виправити. Тому важливо якомога швидше розпочати тестування. Іншим важливим моментом, який слід додати, є безпека. Це, мабуть, найбільш чутлива та найуразливіша частина. Для того, щоб виконати поставлене завдання, веб-додаток повинен працювати, як планувалося. Дотримання вимог продукту є обов'язковим, до певної міри, тому що це допомагає отримати бажані результати.

Кінцевою метою розробленого продукту є надання максимальної задоволеності клієнтів. Причини, через які програми та програмне забезпечення повинні бути перевірені, - це забезпечити найкращий досвід користувача. Бути кращим продуктом на ринку допоможе завоювати надійних клієнтів, які матимуть великі довгострокові ефекти.

ВИСНОВКИ

В ході виконання даної дипломної роботи було проаналізовано предметну область розробки, а саме систему оцінки якості обслуговування закладів харчування. Було розглянуто основні типи клієнт-серверних архітектур, моделі передачі даних, їх переваги та недоліки. Після дослідження та аналізу переваг і недоліків, було зроблено висновок, що для системи оцінки якості обслуговування в закладах харчуваннях є доцільним обрання монолітний тип архітектури. Монолітна архітектура зручна для роботи з маленькими командами, тому було вибрано такий підхід для створення дипломного проекту. Вона повністю покрила всі вимоги для дипломного проекту. Компоненти монолітного програмного забезпечення є взаємопов'язані і взаємозалежні, що допомагає програмному забезпеченню бути автономним. Для моделі передачі даних було обрано архітектурний стиль REST. Він забезпечує чудову продуктивність, зокрема, через кешування інформації, яка не змінена і не динамічна.

Було розглянуто існуючі рішення на світовому ринку оцінки закладів харчувань. Вибрано сильні сторони конкурентів та реалізовано в дипломному проект

Завдання дипломного проекту з вирішенням проблем пов’язаних з оцінкою та проектування системи відгуків для закладів харчування, дослідження методів зберігання та використання великих об’ємів даних для швидкого та комфортного перегляду користувачем виконано. Розроблено чітку структуру шкали оцінки, зручний інтерфейс веб-додатку та впроваджено гнучку систему коментарів. Головною унікальною ознакою системи повинен є механізм обробки інформації, що генерує текстовий файл, який повністю готовий для друку. Можливо на регулярній основі друкувати власні книги про заклади харчування та усією інформацією щодо їх якості. Система має локалізацію за країною, щось схоже на “франшизу”. Кожна країна може мати таку систему з відповідним доменом, який вказує на локалізацію. Система є повністю

автоматизованою і зручною в налаштуванні. Так група головних редакторів може легко редагувати і заповняти дані про той чи інший заклад харчування. Щоб оцінка якості закладів харчування не була доволі суб’єктивною та не базувалася на думці одного користувача, система має два головних критерія оцінки якості закладів харчування: оцінка на основі коментарів та рейтингу користувачів системи, оцінка на основі незалежних експертів. Також кожен користувач має змогу знайти повну інформацію про той чи інший заклад харчування та його місце розташування. Для кращого підбору закладів харчування повинна було сформовано сторінку веб-додатку з динамічно заповненою картою, користувач може відсортовувати результат пошуку за заданими параметрами та за місцем розташуванням.

Система покрита тестами на 80% відсотків, бо кінцевою метою розробленого продукту є надання максимальної задоволеності клієнтів. це допоможе бути кращим продуктом на ринку та завоювати надійних клієнтів, які матимуть великі довгострокові ефекти.

ПЕРЕЛІК ПОСИЛАНЬ

1. Client–server model [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Client-server_model.
2. Model-View-Controller [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Model-View-Controller.
3. Software Testing Overview [Електронний ресурс] – Режим доступу до ресурсу:https://www.tutorialspoint.com/software_engineering/software_testing_overview.html.
4. Best Architecture for an MVP: Monolith, SOA, Microservices, or Serverless? [Електронний ресурс] – Режим доступу до ресурсу: https://rubygarage.org/blog/monolith-soa-microservices-serverless.
5. About MySQL [Електронний ресурс] – Режим доступу до ресурсу: https://www.mysql.com/about/.
6. An Introduction to JavaScript [Електронний ресурс] – Режим доступу до ресурсу: https://javascript.info/intro.
7. About Ruby [Електронний ресурс] – Режим доступу до ресурсу: https://www.ruby-lang.org/en/about/.
8. WHAT IS A GEM? [Електронний ресурс] – Режим доступу до ресурсу: https://guides.rubygems.org/what-is-a-gem/.
9. Ruby on Rails 2.1 - Unit Testing [Електронний ресурс] – Режим доступу до ресурсу: https://www.tutorialspoint.com/ruby-on-rails-2.1/rails-unit-testing.htm.
10. Integration Testing: What is, Types, Top Down & Bottom Up Example [Електронний ресурс] – Режим доступу до ресурсу: https://www.guru99.com/integration-testing.html.

Додаток А. Структурна схема варіантів використань

Додаток Б. Схема структурна станів системи

Додаток В. Схема бази даних